Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Security Monitoring and Alert Correlation for Network Intrusion Detection

Dissertation zur Erlangung des Doktorgrades an der
Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik
IT-Sicherheit und -Sicherheitsmanagement
der Universität Hamburg

vorgelegt von Steffen Haas
geb. 1992 in Mainz

eingereicht am 30.10.2020

Betreuer: Prof. Dr. Mathias Fischer

Erstgutachter: Prof. Dr. Mathias Fischer

Zweitgutachter: Prof. Dr. Michael Meier

Tag der Disputation: 04.03.2021

Zusammenfassung

Angriffe auf IT-Systeme können zu erheblichen Störungen mit netzwerkweiten Auswirkungen führen. Standardmäßig basiert die Angriffserkennung auf einem Intrusion Detection System (IDS). Ein Security Operations Center (SOC) kann die Menge an resultierenden Alerts jedoch nicht analysieren. Aufgrund dieser Flut an Alerts, werden die wirklich gefährlichen Angriffe mit wenigen Alerts, wie die Advanced Persistent Threats (APTs), nicht erkannt. Unternehmen setzen als Lösung Security Information and Event Management (SIEM) Systeme zur Korrelation von Alerts und anderen sicherheitsrelevanten Daten ein. Diese Systeme fassen jedoch meistens nur den Sicherheitsstatus der IT-Systeme zusammen. Eine Priorisierung der Alerts und ein Kenntlichmachen der darin enthaltenen Angriffe erfolgt damit nicht. Dies wird zusätzlich durch die beschränkte Sichtbarkeit der oft eingesetzen Netzwerk-basierten Intrusion Detection Systeme (NIDSes) erschwert. Grund dafür ist, dass sich nicht alle Angriffsaspekte im Netzwerkverkehr manifestieren und der interne Netzwerkverkehr dem NIDS verborgen bleibt.

Diese Dissertation präsentiert Mechanismen für eine umfassende Erkennung und Rekonstruktion von Angriffen. Die Beiträge setzen auf zwei unterschiedlichen Ebenen an. Zum einen produziert das Sicherheitsmonitoring qualitativ hochwertige Daten, die dann zur Erstellung von Alerts genutzt werden können. Zum anderen zeigen Korrelationsmechanismen Zusammenhänge zwischen Alerts auf und fassen die rekonstruierten Angriffe zusammen. Ganz konkret verbindet eine gemeinsame Host- und Netzwerküberwachung entsprechende Daten in Echtzeit. Eine erweiterte Sichtbarkeit wird dabei durch die Attributierung von Netzwerkflüssen zu Hostapplikationen erreicht. Die Korrelation von Netzwerkflüssen untereinander ermöglicht außerdem eine robuste Erkennung von verteilten Angriffen auch bei eingeschränkter Sichtbarkeit. Die vorgeschlagene Korrelation von Alerts unterscheidet zwischen den Alerts von Massenangriffen und den weniger häufig auftretenden Alerts von APTs. Nach dem Zusammenbringen und Aggregieren von Alerts des gleichen Angriffs, werden die Angriffe weiter korreliert, um die Angriffsschritte und deren Zusammenhänge hervorzuheben. Für die Skalierbarkeit in großen Netzen setzen die vorgestellten Mechanismen auf einer verteilten Überwachungs- und Korrelationsplatform auf, was bei entsprechendem Einsatz zu einer flächendeckenden Angriffserkennung führt. Der Betrieb als Collaborative Intrusion Detection System (CIDS) wird durch einen weiteren Mechanismus ermöglicht, der Zusammenfassungen von Alerts für deren Austausch erstellt.

Die entwickelten Ansätze wurden umfassend individuell als auch in Kombination auf der Basis von realem Einsatz, Testumgebung und Simulation evaluiert. Weiterhin wurden die Beiträge zusammen entlang einer bestimmten Abfolge diskutiert. Das Gesamtsystem erkennt Angriffe mit hoher Genauigkeit durch die Korrelation verschiedener Informationen. In einem realen Einsatz war die Attributierung bei mehr als 96% der TCP Verbindungen erfolgreich. In einer realistischen Simulation wurde ein Peer-to-Peer (P2P) Botnetz auch dann robust erkannt, wenn die NetFlows den Netzwerkverkehr von nur 5% der Bots abbilden. Zugleich fasst das System diese, das gesamte Netzwerk bedrohende, Angriffe zusammen. Echte Alerts ließen sich in Experimenten durch Aggregation auf 0,6% der ursprünglichen Anzahl verringern. Beim Einsatz des Systems als CIDS, führten die Zusammenfassungen von Alerts zu einer Reduktion des Datenverkehrs beim Austausch auf etwa 1% der Originaldaten.

Abstract

Attacks on IT systems can have network-wide impacts with tremendous consequences. For attack detection, the standard solution is to deploy an intrusion detection system (IDS). However, it reports too many alerts to be all analyzed by the security operations center (SOC), even with the help of alert correlation. This creates alert fatigue and the really sophisticated attacks, the advanced persistent threats (APTs), that trigger only a few inconspicuous alerts, go unnoticed. To mitigate the alert correlation problems, companies utilize security tools like security information and event management (SIEM) systems that correlate alerts and other security-relevant data. Although these systems provide extensive data analytics, they just summarize the overall security status of IT systems but fail to appropriately prioritize alerts and to make attacks in the alert data visible. A solution to achieve this is additionally impeded by the restricted visibility of the commonly deployed network intrusion detection systems (NIDSes), because not all attack aspects manifest in the network traffic. Furthermore, the NIDS location enables to capture the Internet traffic of the monitored network but not the traffic between hosts inside the network.

This dissertation presents mechanisms that enable a comprehensive detection and reconstruction of attacks. The novel contributions work towards a better overall detection accuracy at two different stages of the intrusion detection process. First, security monitoring is enhanced to produce high-quality monitoring data and to leverage it for an accurate reporting of alerts. Second, novel alert correlation mechanisms identify relations among the alerts and summarize the reconstructed attacks. In particular, a joint monitoring of hosts and the network correlates respective monitoring data in real-time. An extended visibility is established through the attribution of network flows to host processes. In addition, detectors leverage correlated network flows to robustly detect the characteristics of some distributed attacks despite restricted visibility in the monitoring data. The proposed alert correlation separates alerts belonging to high-volume attacks from the infrequent, i.e., spatially and temporally dispersed, alerts belonging to a stealthy APT. After aggregating and bringing together alerts of the same attack, they are correlated to reconstruct the attacks, highlighting the performed steps and how they interconnect. To scale with large networks, the proposed mechanisms integrate into a distributed monitoring and correlation platform that allows comprehensive intrusion detection when deployed to several locations inside the network. The deployment as a collaborative intrusion detection system (CIDS) is supported by another mechanism that efficiently exchanges summaries of alerts.

The developed approaches have been extensively evaluated individually and in combination with each other on the basis of real-world deployments, testbeds, and simulations. Furthermore, the contributions are discussed altogether along a detection pipeline. The overall system accurately detects attacks by correlating a variety of information. It achieved a real-time attribution for more than 96% of TCP connections in a real-world deployment. In realistic simulations, a peer-to-peer (P2P) botnet was detected robustly, as NetFlows covering the traffic from only 5% of the bots were sufficient. At the same time, the concise attack summary highlights attacks that threaten the network at large. Alert correlation experiments condensed real-world alerts into aggregations that correspond to 0.6% of the original amount of alerts. Using alert summaries reduced the exchange volume in a CIDS to about 1% of the full alert data.

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich auf meinem Weg zur Promotion begleitet und auf unterschiedliche Weise unterstützt haben.

Für die intensive Betreuung dieser Arbeit bin ich meinem Doktorvater Mathias Fischer sehr dankbar. Seine durchgehende Unterstützung in Form von Anmerkungen und Diskussionen hat sowohl die Suche und Aufarbeitung meines Dissertationsthemas, als auch die Konzeptionierung und Ausarbeitung meiner Forschungsbeiträge sowie diese Doktorarbeit maßgeblich geprägt.

Darüber hinaus möchte ich meinen Kolleginnen und Kollegen der Arbeitsgruppe IT-Sicherheit und Sicherheitsmanagement sowie des gesamten Arbeitsbereichs Security and Privacy an der Universität Hamburg danken. Auch besonders aufgrund des kollegialen, sozialen und herzlichen Arbeitsklimas kann ich auf eine großartige Doktorandenzeit zurückblicken. Nicht nur bin ich dafür dankbar, in meinen Kolleginnen und Kollegen neue Freunde gefunden zu haben, auch gebührt ihnen Dank für ihre Unterstützung meiner Arbeit. Insbesondere möchte ich Florian Wilkens für den regen Austausch von Forschungsideen und die Zusammenarbeit an gemeinsamen Publikationen danken. Auch für die thematische Überschneidung mit Matthias Marx und die daraus resultierenden Diskussionen bin ich sehr dankbar. Ebenso danke ich David Jost für die unzähligen Male, die er mir unermüdlich beim Fehlersuchen und bei allerlei technischen Problemen geholfen hat. Nicht unerwähnt lassen, möchte ich auch Doganalp Ergenc, der stets für eine gelassene und gleichzeitig motivierende Stimmung in unserem Büro gesorgt hat.

Neben allen weiteren Kolleginnen und Kollegen, sowie Korrekturlesenden dieser Arbeit, gilt mein Dank Robin Sommer für die langjährige Zusammenarbeit an dem Zeek IDS und das Einbringen seiner praktischen Erfahrung in meine Forschung. Ebenso haben mir die Untersuchungen zu Cyberkriminalität im Bereich Botnetze wertvolle Einblicke vermittelt, die zwar keinen Einzug in meine Dissertation gehalten haben, für die ich allen Beteiligten dennoch sehr dankbar bin.

Dass ich überhaupt meinen Weg zur Promotion gehen und diese abschließen konnte, ist nicht zuletzt meiner Familie zu verdanken – meinen Eltern, meinem Bruder und meinen Großeltern. Sie haben gewiss den größten Anteil daran, was ich in meinem Leben erreicht habe. Während der Promotionszeit war ich mal mehr, mal noch weniger als sonst für meine Familie, aber auch Freunde erreichbar. Dennoch hat mir jeder Kontakt mit Freunden und Familie Energie und Ermutigung gegeben, um in dieser fordernden Zeit bis zum Schluss durchzuhalten. Exemplarisch dafür bedanke ich mich bei meiner Mutter Ines Haas für die vielen Telefonate. Zu guter Letzt möchte ich meiner Partnerin Stephanie Schaffert danken, die wohl mit Abstand das meiste Verständnis während meiner Promotionszeit und insbesondere der Schreibphase aufbringen musste und mir trotzdem jederzeit den Rücken freigehalten hat.

Contents

1 Introduction and Motivation

Since IT systems often provide critical services or hold valuable information in our digitized society, these system have become a lucrative target for criminals. However, storing and processing all these valuable information is no longer performed in isolated environments with dedicated IT systems. Instead, services and data processing are implemented by IT systems that are connected in large networks or even globally over the Internet. Thus, Internet or network communication in general has become the backbone of our digitized society.

Recently, cybercriminals have learned that for achieving their goals, it might require them to gain access to the organization's network and thereby enabling access to the critical services and valuable information the criminals target at. Especially the attacks aiming on the network at large can have tremendous consequences for the targeted organization. Attackers advancing this way are usually sophisticated enough to achieve their goal like dumping the customer database with credit card information or spying on business secrets. This is already causing million dollars of loss in revenue [Acc19]. Furthermore, the consequences of such attacks are also that the network is left in an insecure state as the attackers are likely to still have control over the network after achieving their goal. While a secure state might be restored with ease after an attack with only very local and limited consequences, restoring the state of a network that is affected at large is not that easy. Apart from the technical effort to clean all compromised systems, the service and network operation is probably disrupted either already by the attack itself or while recovering from it when systems have to be taken offline temporarily. This puts additional burden on keeping the business and its critical processes running, because nowadays almost all processes rely on IT services in the network and will be affected during a network-wide attack or while recovering from it by some means or other.

To protect the network and its communicating hosts, it is common practice to deploy intrusion detection systems (IDSes) that detect malicious events and report them as alerts to make the security operations center (SOC) aware of an intrusion. The most established detection techniques to classify the monitored events of a supervised system are signature-based and anomaly-based intrusion detection, which identify known malicious events or unexpected events, respectively. In case of the regularly deployed network intrusion detection systems (NIDSes), the detection system is monitoring and analyzing the network traffic for malicious communication of an attacker. For simple and very local attacks targeting only a single host in the network, singleton alerts might already enable the SOC to understand the relevant aspects of the attacks and to initiate effective countermeasures. For larger attacks or those that trigger more alerts, alert correlation exists to highlight the relations among alerts of the same attack. The correlated alerts reflecting the collection of malicious events of a larger attack, enable the SOC to also understand these attacks and to mitigate them appropriately.

1.1 Problem Statement

Highly sophisticated attacks, such as advanced persistent threats (APTs) [CDH14], try to evade intrusion detection as much as possible in the first place, leaving the SOC with potentially only few alerts as indicator for the attack. Understanding these alarming indicators and reacting upon them is essential to prevent the attack from turning into a severe security breach. Thus, the detection of attacks targeting a network at large must happen as soon as possible to prevent the attackers from steadying control and succeeding with next steps towards their goal [HCA11]. However, the SOC can rarely keep up with the regular flood of alerts from various intrusion attempts every day, resulting in so many false positives as well. This is causing alert fatigue with the consequences of the most important alerts not being prioritized, their relations among each other not being unveiled, or the few traces of the sophisticated attacker not being noticed at all. Central research questions must therefore lie in developing mechanisms that allow to equip the SOC with the right tools to confront sophisticated attackers targeting the network at large.

Research Question Q1: *How can complementing monitoring sources be utilized for network monitoring in real-time to compensate restricted visibility?*
To fully reconstruct an attack, the respective activity must be captured through monitoring in the first place. However, some attack aspects might stay hidden from a regular network monitor and NIDS because of restricted visibility resulting from the attack traces not becoming manifest in the network traffic. Furthermore, network traffic tends to become encrypted for security and privacy reasons [KDH16], and also malware is already encrypting and self-modifying its code to evade detection [BLS13]. For the purpose of collecting and correlating logs from different sources, security information and event management (SIEM) systems [WN05; BMZ14] are commonly used in practice. However, besides host logs, SIEM systems usually consume only communication summaries and network statistics but have no visibility into the traffic. Therefore, the log data but also the correlation mechanisms are too coarse-grained to allow the detection of sophisticated attacks.

Research Question Q2: *By which mechanisms and to which extent can correlated monitoring data be utilized to detect network- or Internet-wide attacks thoroughly with all their related activities?*
In best case, the activity of an attacker is obviously malicious from the perspective of an IDS. However, the classification of singleton events to detect malicious ones might not always be possible if the malicious event itself cannot be distinguished from a benign one. Especially in distributed network attacks [ZLK10] where a lot of hosts are involved, each one is communicating with some of the others. While a singleton pair of communicating hosts might look inconspicuous, the view on correlated pairs of communicating hosts might reveal the malicious intent of the communication altogether.

Research Question Q3: *How can low-level security alerts from an IDS be aggregated and correlated to concisely summarize the intrusion detection result, highlighting especially network-wide attacks like distributed or multi-step attacks?*
For any malicious communication that is detected by a NIDS, an alert is reported to the SOC. However, a singleton alert does not necessarily reflect the full attack, as the attacker probably interacts with the targeted network several times and performs several different actions to pursue

a particular goal. Thus, reconstructing an attack means to identify related alerts and to assemble them to a picture of the attack in its entirety. Alert correlation algorithms usually assist the SOC in this task. However, presenting the collection of related alerts is not sufficient to make the SOC develop an understanding of the attack. Instead, the alert correlation result should to highlight certain characteristics of the attacks to allow their understanding and mitigation. Especially attacks against many hosts at once potentially with coordinated activity from multiple sources [ZLK10] as well as attacks that target hosts consecutively for lateral movement [Boh+17] have to be reconstructed carefully. Otherwise, the attack is likely to be underestimated and mitigated inappropriately.

Research Question Q4: *How can alert correlation mechanisms correlate temporally and spatially distributed alerts from APT attacks?*
Especially stealthy attacks [Rud+16] and APTs [CDH14] impede the reconstruction of attacks from alerts, because they have their activities obfuscated by long time periods in between consecutive activities. Detecting such attacks is especially difficult because the time between respective activities is filled with unrelated malicious activities or simply with false positives in terms of noise from the Internet [BGD17; Son+11; BCF12]. When the SOC is considering only recently reported alerts for alert correlation, the outcome can never bring together a new APT alert with the previous one that already occurred quite some time ago in the past. However, correlating new alerts with any other alert that ever occurred in the past is neither reasonable nor computational feasible. Consequently, APT alerts are likely to never be correlated and to attract the attention from the SOC that would be required to prioritize and handle them appropriately.

Research Question Q5: *Which benefits in terms of volume savings is provided by exchanging summaries of alerts that are reported locally by distributed IDS sensors in the network, and what are appropriate attack representations to enable further collaborative analysis?*
While network sizes and also the traffic volume grow, a single NIDS is not capable anymore to monitor and analyze all traffic centrally. For scalability reasons, multiple sensors are distributed in the network and work together in a so-called collaborative intrusion detection system (CIDS) [Vas+15b]. In contrast to standalone IDSes, the IDS sensors in a CIDS exchange data among each other to collaboratively perform intrusion detection and alert correlation. A fully distributed CIDS without a central processing node avoids a single point of failure (SPOF) but does not scale well when every sensor exchanges the full local data with any other sensor. With respect to alert correlation, every sensor would receive all alerts from every other sensor to potentially identify a small fraction of received alerts that correlates with some local alerts. However, when parts of a network-wide attack are captured by several sensors, a more coarse-grained exchange of alert data could save bandwidth and processing power while still identifying alerts that are required for assembling the big picture of the network-wide attack.

The next section briefly summarizes the answers to the questions provided by this thesis.

1.2 Contributions

Figure 1.1 illustrates how the contributions of this thesis, as described in the paragraphs given below, are pieced together to solve the challenges of Section 1.1 during monitoring, detection, and correlation to detect malicious events and summarize them in an attack report. Based on

the working of a traditional NIDS, *zeek-osquery* extends the monitoring visibility through a joint monitoring that correlates host and network events in real-time and provides the correlation result for intrusion detection. In addition to the malicious communication that an NIDS already detects, the detection of *scan campaigns* and *peer-to-peer (P2P) botnets* demonstrates how intrusion detection can leverage the correlated events using the example of correlating the hosts' communication relations. Based on all the network alerts that an IDS reports regularly, *graph-based alert correlation (GAC)* clusters alerts to attacks and links them to multi-step attacks in the final intrusion summary. *Weak alert correlation* complements the clustering by assembling the temporally dispersed alerts of stealthy attack steps over long time periods. *Collaborative attack correlation* complements the linking of attacks and their steps by enabling the efficient identification of network-wide attacks that are captured by multiple NIDSes distributed in the network.

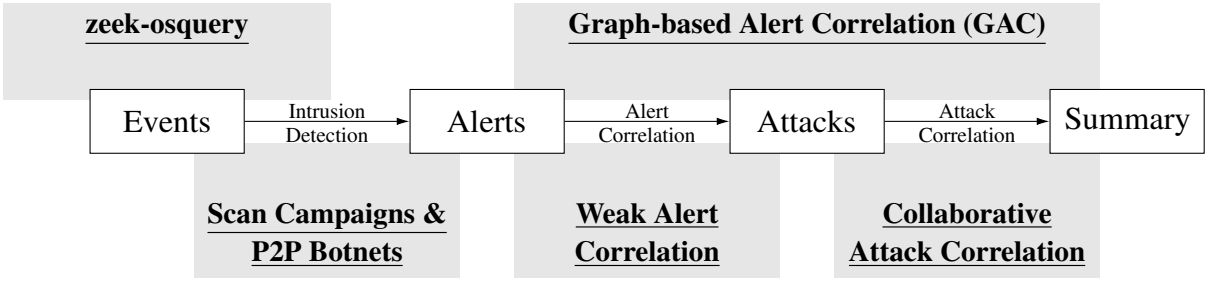


Figure 1.1: Relations between the contributions of this thesis.

Fine-grained Visibility into Networks and Host Communication Restricted visibility is a major problem for intrusion detection and threat hunters. Both require high-quality monitoring data to detect, investigate, and reconstruct a security incident. However, the traces of an intrusion found in network communication cannot unveil the host semantics or consequences of the malicious communication. Thus, network monitoring and network intrusion detection can often indicate an attack but are blind when it comes to capturing the circumstances of the attack. Furthermore, the current trend towards traffic encryption [For18] causes the visibility of a NIDS to become even worse. As an immediate countermeasure, Transport Layer Security (TLS)[1] Proxies [ONe+16] restore visibility specifically into encrypted communication, but come with several drawbacks [WMY18], including the violation of end-to-end encryption and privacy. To further extend the visibility, SIEM systems [BMZ14] try to compensate the restricted visibility of a NIDS by correlating monitoring data from several logs. Their correlation, however, is too inaccurate because it is based on log timestamps, the log files are too coarse-grained, an interactive retrieval of additional data is not possible, and the system itself is not scalable.

To mitigate the negative effects of restricted network visibility on the detection accuracy, this thesis revises the idea of an early work from Snapp et al. [Sna+91] and enhances the monitoring quality through extended visibility. Their idea of correlating data from network and host monitors is extended and generalized for broad intrusion detection purposes. The resulting concept has been published in [HSF20], including the open-source prototype *zeek-osquery* that demonstrates the benefits arising from linking network communication to process-centric host activity in real-time. The benefits from this fine-grained host-network correlation include a

1. Unless referring to a specific version, the protocols TLS and its outdated predecessor Secure Sockets Layer (SSL) are used interchangeably.

joint visibility on hosts and the network through the correlation of monitored network traffic by host context. Because of the real-time correlation of the data during monitoring, the NIDS Zeek [Pax99] can directly make use of the additional host context when assessing the network communication. The experiment results for attributing network flows to the originating process on a host demonstrate that zeek-osquery gives valuable insights into the host semantics for the hosts's network communication.

Collective Monitoring for the Detection of Internet-wide Attack Scenarios The real intend of some malicious activities cannot be assessed by classifying them individually because they cannot be differentiated easily from benign behavior. However, especially Internet-wide attacks, i.e., attacks that target multiple network sites simultaneously, leave many traces that together can unveil the attack scenario. To identify traces that belong to an attack scenario with many hosts involved, alert aggregation can be performed to highlight predominating alert features such as IP address or ports. The presence of a predominant feature might serve as an indicator for coordinated activities [ZLK10] like in distributed denial-of-service (DDoS) attacks [MMS16], port scans [BDA13; BBK11], worm spreading [KS14], and command and control (C2) communication of botnets [GZC14]. Most existing approaches that leverage alert aggregation to detect such scenarios require a certain level of similarity among the alerts, e.g., for the IP address. However, this is not always given for attack scenarios with different sources and targets [Dai+15].

In contrast to relying on a predominant feature that indicates a coordinated attack, this thesis leverages characteristics that are specific to particular attack scenarios. The principle for their detection is to find network activities that together reflect scenario-specific characteristics. In the context of this thesis, scan campaigns and P2P botnets are chosen as examples for Internet-wide attack scenarios with special characteristics in the network communication. The detection of these two scenarios has been published in [HWF20] and [Muh+18]. Besides traditional byte-matching signatures, these examples demonstrate scenario-characteristic signatures that require the collective assessing of network communication instead of individual network activities. The experiment results indicate that scan campaigns and P2P botnets with Internet-wide scope can generally be detected by the proposed scenario detection even if only locally deployed at one of the targeted network sites. For better detection accuracy, multiple network sites can collaborate in a collective monitoring to capture a larger fraction of the Internet traffic.

Alert Correlation for the Detection and Reconstruction of Network-wide Attacks The detection of attacks is usually based on an IDS that detects low-level attack indicators like byte patterns and reports them as alerts. However, this generates a huge amount of alerts every day, including also false positive and low-priority alerts. Without further processing the alerts and describing them in the picture of an attack, insufficient mitigation actions might be chosen. Especially distributed attacks result in many alerts, but an intrusion summary of such an attack should actually include only the most important information among all the alerts' details. Similarly for multi-step attacks, an intrusion summary should explain the linking between consecutive attack steps with only the important information. Thus, aggregating and filtering during alert correlation must be performed carefully regarding the alert details, such that relations among alerts stay apparent and that the intrusion summary holds the most valuable information. Otherwise, the detection result would be small and independent attacks, and the SOC cannot see the real network-wide attack that the attack pieces actually form.

To achieve the goal of detecting network-wide attacks and condensing respective alerts to a concise representation, this thesis proposes an alert correlation algorithm for assembling alerts to attacks that works in two steps: (1) identifying alerts that all describe the same action of the attacker and then (2) linking the alerts of consecutive actions that build upon each other to carry out the attack. A graph-based implementation of this approach named *graph-based alert correlation (GAC)* has been published in [HF18]. In addition, this approach makes use of supplementing scenario-context that is derived from the alerts themselves and incorporated into the linking of consecutive attack steps. This has also been published in [HF19]. In summary, the result of this alert correlation is a reduction of alert volume by summarizing the attack actions into an intrusion report that especially highlights the links among the actions of a multi-step attack. Experiments on real-world data demonstrate that up to 99.3% of the alerts can be summarized this way.

Continuous Alert Correlation over Long Time Periods to Support APT Detection Alert correlation has to process a lot of alerts every day and probably fails to link alerts from slow and stealthy attacks because these alerts' weak relations get lost in the shuffle. While some noisy attacks trigger a lot of alerts in a short time period, APT-like attacks trigger only few alerts over a long time period. As a result, regular alert correlation algorithms overlook the relation among the temporally distributed alerts of such slow attacks. This is because most alert correlation algorithms operate on finite sets of alerts, e.g., the last 1000 alerts or those from the last hour. Each of these alert batches is correlated separately and, therefore, the relevant alerts are not correlated across different batches. Also a reapplication of the correlation algorithm to the results across alerts batches fails, because the respective alerts have probably been filtered by then.

Based on these insights, this thesis presents an addition to regular alert correlation algorithms that operate on alert batches. Across these batches, the addition named *weak alert correlation* continuously identifies and aggregates alerts from slow but on-going attacks. This aggregation goes on, until the alerts assemble to an attack step that fits into the context of an APT-like attack. The continuous alert aggregation can supplement other correlation algorithms by providing the alerts of a slow attack step that would otherwise go unnoticed. The experiments regarding an APT attack with multiple steps indicate that this continuous alert aggregations can indeed highlight the stealthy attacker actions.

Exchange of Attack Data for Collaborative Intrusion Detection Large networks often have the problem that their several subnets, remote office branches, and Internet upstreams cannot be monitored centrally by a single NIDS. Instead, these networks deploy multiple IDS sensors that work together in a so-called CIDS [Vas+15b]. Intuitively, the intention is to combine the information from the distributed IDS sensors. The benefit is that an alert reported by a particular sensor can be set in relation to alerts from other sensors to get aware of network-wide attacks that become manifest in several alerts across the different sensors. However, not all alerts and their full details are relevant in this regard, both for efficiency and privacy reasons. A common solution is to exchange alert summaries among the CIDS nodes, either particular features like the attacking IP addresses [Loc+05] or feature aggregations [ZLK09]. Correlating such kind of exchange data in a CIDS, however, requires that related alerts from different sensors have some alert feature values in common in the first place. An attacker can easily circumvent

this detection mechanism when using different source IPs for attacking each supervised network segment.

To efficiently find similarities among the alerts from different sensors independently from the feature values, this thesis defines a compact data structure that characterizes detected attacks, enabling a *collaborative attack correlation*. Instead of exchanging all alerts or alert summaries with concrete values, this attack characterization allows to identify similar attacks based on the attack type, i.e., the attack scenario. This characterization reflects the who is attacking whom structure and has been published in [HWF19]. Compared to exchanging full alert data, the exchanged data volume can be reduced to 1% by using the compact data structure holding the attack characteristics. If two attacks from different sites are identified to potentially be equal, their related alerts can be exchanged in the aftermath to process them with further correlation algorithms.

1.3 Outline

The remainder of this thesis is structured as follows: The background in **Chapter 2** gives an overview on intrusion detection techniques to classify malicious behavior and to detect attacks. In addition, it presents classifications of both intrusion detection and alert correlation algorithms.

Chapter 3 states requirements for IDSes. Apart from that, the intrusion detection process is presented with a detailed description of the tasks required for implementing an effective intrusion detection. Furthermore, the chapter summarizes the state of the art along the process and discusses all related work with respect to the IDS requirements.

In **Chapter 4** this thesis introduces measures for security monitoring that solve several problems that occur at early stages in the intrusion detection process. The goal of these measures is to achieve high-quality monitoring data, to accurately detect malicious behavior, and to report it as alerts. The foundation for the enhanced security monitoring is the correlation of monitoring data, based either solely on network monitoring or on a combination of host and network monitoring.

Chapter 5 presents alert correlation algorithms that are applicable in stages towards the end of the intrusion detection process. These algorithms process IDS alerts and return representations of attacks with a focus on network-wide intrusions. The goal of these algorithms is not only to detect the attacks in question, but also to overcome the additional challenges that arise from temporally and spatially dispersed alerts.

The evaluation in **Chapter 6** assembles the contributions along the intrusion detection process in an end-to-end evaluation pipeline to demonstrate their working and relation among each other. Different data sets and evaluation methodologies are used to conduct the experiments and to evaluate contributions individually and in combination.

Chapter 7 concludes this thesis with a summary and an outlook for future work.

Where to find the concrete contributions and their evaluation in this thesis is illustrated in Figure 1.2 regarding the research questions Q1-Q5 from Section 1.1.

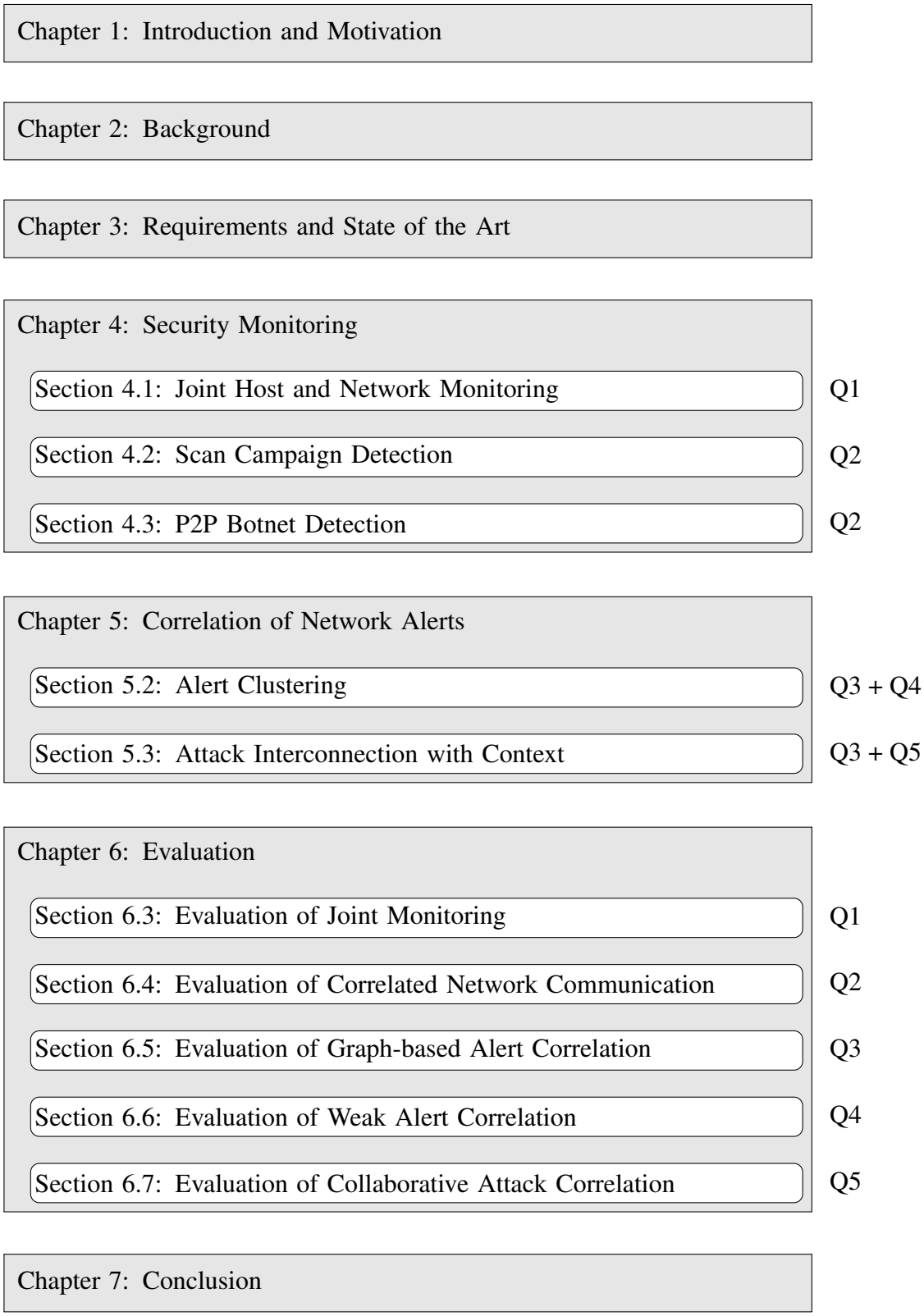


Figure 1.2: Structure of this thesis and sections where to find contributions to particular research questions.

2 Background

This chapter provides essential background information for this thesis. Section 2.1 sketches the communication of hosts in IP-based computer networks and summarizes attacks that target the network as a whole. Section 2.2 describes the fundamentals of intrusion detection in general and highlights the challenges in intrusion detection. Section 2.2 provides a classification of countermeasures to detect intrusions in the network. Section 2.4 describes the security monitoring and intrusion detection tools Zeek and osquery.

2.1 Computer Networks

The following explains the foundations of computer networks that enable communication among the hosts and also the Internet. Afterwards, an overview of attacks are given that an attacker can perform when infiltrating and taking over a network.

2.1.1 Network Infrastructure and Topology

Computer networks nowadays became quite complex because they serve so many needs in our digitized society. With respect to commercial aspects, the computer network connects IT systems and their components among each other to support a business process in the company. Apart from printers, domain-specific devices such as robots in manufacturing industry, or supervisory control and data acquisition (SCADA) systems more generally, the communicating endpoints in a network are traditionally office systems and application servers. These hosts are supposed to provide or access resources in the own network but also to communicate on the Internet. To achieve compatibility among the communicating hosts, several standards like Internet Protocol (IP), Transmission Control Protocol (TCP), and User Datagram Protocol (UDP) evolved over time.

The term hosts in this context reflects commodity hardware and general purpose computer systems. It includes regular personal computers as found in offices but also servers in data centers. These hosts have in common that they implement the concept of the TCP/IP model [FS11], i.e., the Internet protocol suite. The hosts communicate in the IP-based network by running applications that send and receive messages. For the communication in IP-based networks, usually one of the transport protocols *TCP* or *UDP* is usually used. While UDP is connectionless and best-effort similar to IP itself, the connection-oriented TCP comes with some guarantees and makes use of an explicit three-way handshake [Pos+81] to initiate the communication. Either way, related packets in regular communication transmitted between two endpoints, i.e., IP and port tuples, in the network are denoted as a *flow*. Generally, both endpoints show the intent to participate in the flow. However, the receiver of a packet can also refuse to communicate by aborting the TCP handshake, replying with Internet Control Message Protocol (ICMP) error messages, or simply ignoring the packets. The communication is abstracted by the operating system (OS) on the hosts through the use of sockets as an interface to the network packets.

The network packets are disseminated in the local network segment, i.e., subnet, through switches and forwarded to other hosts beyond the border of the local subnet via routers. Following this concept, a computer network is connected to the Internet through a gateway that is the transit point to other networks on the Internet, i.e., the Internet upstream. All communication between the network and the Internet passes this way via the Internet service provider (ISP). Within larger networks, the hosts are usually organized in subnets with routers and sometimes even network address translation (NAT) in between.

Within a subnet, the communication is completely unrestricted, whereas the communication across subnets is often inhibited and filtered by firewalls. A special subnet of a network is the demilitarized zone (DMZ) in which public services, i.e., the Internet servers of a data center, are located. In such a case, perimeter security distinguishes three zones: the Internet is untrusted, internal subnets of the network are trusted, and the DMZ has a trust level in between. The general security policy is to allow communication that is initiated from a zone with higher trust to zones with lower trust and to block unsolicited communication in opposite direction. The DMZ is an exception of this policy as particular services hosted in this zone are explicitly configured to be accessible from the Internet as well. Most commonly, intrusion detection is performed centrally at the border of the network, i.e., at the Internet upstream.

2.1.2 Attacks on Computer Networks

For an attack to have an effect on the computer network at large, the attack consequences affect either a majority of the hosts in the network or particular hosts that are critical for the overall network operation or business processes, respectively. The attacker is not assumed to initially have access to the network nor to be in possession of any privileged knowledge or rights. Instead, an attack on the computer network can origin from an unprivileged attacker on the Internet. While the attack is going on, the attacker infiltrates the network and takes over more and more control.

Several goals exists that an attacker with such attack on the network at large might want to achieve. Generally, two categories of goals are distinguished: Espionage and Sabotage. Being in control of the network, the attacker can utilize the access to various classified information for espionage. Apart from the data files on hosts affected by the attack, the attacker can potentially eavesdrop in the network for further classified information or to gain more knowledge about the network operation. By the same means of access to the hosts and the network, the attacker is eventually in the position to sabotage the network. Through the unauthorized access, the attacker can violate the integrity of data files or network communication. This resulting consequences on the business processes are tremendous.

To achieve their goals, attackers target the computer network at large with one or a combination of several of the following attacks:

DDoS In a distributed denial-of-service (DDoS) [MMS16] attack, multiple sources cause an overload situation at the target that renders a particular network service unavailable to everyone. The overload is regarding a specific resource, including the network, computational power, or storage capacity. Examples for DDoS attacks are smurf attacks [Kum07] to saturate the network bandwidth, asymmetric DDoS attacks [Che+16] with operations like Transport Layer Security (TLS) handshakes to make the server perform expensive cryptographic calculations, or TCP

SYN flood attacks [Edd07] that cause the number of (half-opened) connections on the server to exceed their limit in memory.

As these attacks scale with the number of attacking sources, the respective resource on the target effectively becomes overloaded once a critical mass of sources is reached. Specifically for most DDoS attacks, the source is not required to receive the response. Instead, spoofing and using multiple source IP addresses gives the target a hard time to detect the attack. In fact, some DDoS attacks even require to spoof the source IP address as in distributed reflective denial-of-service (DRDoS) attacks [Ros14].

Port Scan Networks usually offer a limited set of their services for usage on the Internet, e.g., web or mail servers, remote shell access over telnet or Secure Shell (SSH), or a virtual private network (VPN), among others. Of course, some services require the user to authenticate before being able to proceed. While unauthorized users should not be able to interact with these services, misconfiguration or software bugs enable an attacker to remotely exploit this service.

To determine the attack surface consisting of such insecure services to infiltrate the network, an attacker must know of their existence in the first place. Thus, the attacker searches the network for online hosts and any public services running on them, well-known as port scan [BBK11; BDA13]. Open ports on the hosts indicate a running service, and the port number usually indicates the type of service, e.g., SSH on port 22. Technically, the attacker tries to connect to an IP address and port at which a service is presumed to run. If the attempt fails, no such service is running. However, upon successful connection, the attacker learns that this particular service exists and is accessible from the Internet.

Worm Spreading To efficiently increase the number of infected hosts on the Internet and infiltrated networks, the attacker makes a worm [KS14] spread to any host vulnerable to the particular exploit. Instead of the attacker performing the exploit on every host, the infected hosts themselves automatically try to infect more hosts. One of the first Internet-wide spreading was been seen with the Morris worm in 1988 [Eis+89].

Even though this scalable way of exploiting hosts is an attack itself, the purpose of worm spreading is usually only to infect as many hosts as possible with the respective malware. Once under the control of the attacker, the infected hosts is used for consecutive steps later on. Anyhow, the spreading itself only describes the malware that further replicates to more hosts.

Botnet An attacker that infected a large collection of hosts wants to control them efficiently. Thus, the infected hosts, i.e., bots, are integrated into a botnet and collectively controlled by the attacker, i.e., botmaster [Bil+12]. Thus, the bots establish a channel to the botmaster and await commands to execute. Apart from using all bots for launching DDoS [Sch+10], sending spam mails [Pat+09], or stealing banking credentials [And+13], the botmaster can decide to leverage bots differently when they are identified to run in a network that is of value for the attacker [OBr16].

In recent years, the architecture shifted from centralized botnets to peer-to-peer (P2P) botnets. While all bots connected to a centralized command and control (C2) server [RD13] back then, nowadays the bots interconnect among each other. These bots use their interconnections among each other to propagate new commands from any location in the P2P botnet over multiple hops

to every other bot in the botnet. Instead of the C2 server as a single point of failure (SPOF), the P2P communication makes the botnet resilient to takedown attempts [Haa+16].

Stepping Stone Once the attacker is in control of a host in the target network, subsequent attack actions are performed against the network. To hide the true origin of the attack, i.e., the attacker on the Internet, further actions can be tunneled through this controlled host. Hosts that relay attack traffic this way are denoted as stepping stone hosts [WR10]. Attacks though a stepping stone are especially hard to detect because the exploited host on the internal network is likely to have a more flexible security policy than the stricter policy that applies to Internet hosts.

Characteristic for stepping stone attacks is that the attack traffic is tunneled through the exploited host almost without any modifications. This is particularly true for any interactive attacks, e.g., those that involve telnet or SSH [ZP00]. In such a case, the incoming traffic from the attack to the exploited host equals the outgoing traffic from the exploited host to the actual target.

Lateral Movement Another form of abusing an internal host under control of the attacker is lateral movement [Boh+17]. In this kind of attack, the attacker cannot access the actual target host directly because of topology or security restrictions in the network. Thus, the attacker infects hosts on a path towards the target host.

In contrast to stepping stone attacks that cover up the attack origin, the intention of lateral movement is to access hosts on the target network that otherwise would be unavailable to the attacker. While in a stepping stone attack each host on the path would just relay the attack traffic, exploited hosts along the path of lateral movement eventually establish a direct channel to the attacker once under the control of the attacker.

2.2 Intrusion Detection

Detecting intrusions is necessary in the first place, because full protection of IT systems is not possible. One reasons for that is security management accepting the risk because appropriate measures are found to be too expensive or too inflexible [Sch92]. Even when trying to mitigate all risks, wrong postulates about the attacker capabilities leave unforeseen threats unhandled. The example of so-called nation state hackers illustrate the imbalance between the attacker's almost unlimited resources and the economic reasonableness for security decisions of enterprises. Apart from that, vulnerabilities in the IT systems might remain unpatched because of compliance reasons or because they are simply unknown to the public.

To improve the overall IT security situation, IT security management often defines a defense strategy that includes prevention, detection, response, and recovery (PDRR) as a cycle. The detection part of this cycle is implemented by several security tools. Apart from an intrusion detection system (IDS), additional tools like honeypots, an intrusion prevention system (IPS), or a security information and event management (SIEM) system might be used.

2.2.1 Intrusion Detection Goal and Basics

For the detection of attacks, the IDS supervises computer systems and the communication infrastructure for signs of intrusions and misuse. Similarly, an IPS detects the attacks and additionally initiates countermeasures to stop the attack. The benefits for the overall security situation of having an IDS in place becomes apparent in the picture of the PDRR cycle. A successful detection of the attack and attacker, respectively, enables the initiation of attack response mechanisms to limit the attack damage. Beyond that, knowledge about the security incident is gained when reconstructing the attack and recovering from it. This knowledge allows to improve on the preventive measures.

Intrusion detection starts with recording all security relevant events of the supervised system. In general, *monitoring* the system this ways audits operational information, including the accessed resource, who accessed it, at which time it was accessed, and how the resource was accessed. The recorded audit data is specific to the type of monitored system. Examples for data an a computer include the opening of files or the execution of programs, whereas examples for data in the network include connection establishments and releases as well as packets that are transmitted between specific systems. Furthermore, also application specific logs exist that detail audit the internals at application runtime. Anyhow, the integrity of audit data must be ensured, or otherwise the attacker can tamper with it to cover up the attack traces. The recorded audit data collects the input of intrusion detection.

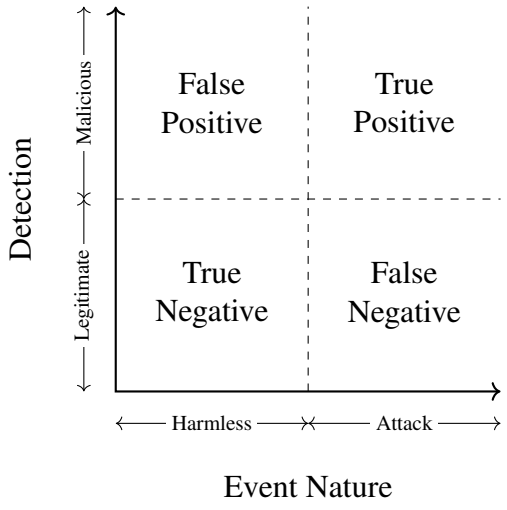


Figure 2.1: Classification of events for intrusion detection.

Intrusion detection is an automatic analysis of this audit data, i.e., monitoring logs. The basic idea of the analysis is to *classify* the recorded events. An event is the result of either a legitimate and harmless use of the supervised system or of an attack. Thus, the analysis is supposed to classify events into those that are legitimate and those that are malicious. Figure 2.1 illustrates the classification of events. The nature of the events is either harmless or an attack. The detection, i.e., classification, tries to identify the nature and labels the event with legitimate or malicious, respectively. A positive detection reflects an events that the classification labels as malicious, independent from the actual event nature. However, errors in classifying the events can occur. Apart from the correct classification, i.e., true positive and true negative, the classification result can be wrong, i.e., false positive and false negative.

The two major classes of classification mechanisms used for the detection are based on signatures and anomalies, respectively. The basic idea of *signature* detection is that attack patterns can be described in sufficient detail. A positive detection requires the event to match such an attack signature. Such signatures are specified as rules that describe the attack specifics. The drawbacks of the signature-based detection are the required prior knowledge about the attacks and that new attacks require to update the signature database. An example for a signature is the specific byte sequence in the payload of a network message sent by a particular malware. In contrast, the basic idea of *anomaly* detection is that attack behavior statistically differs from other behavior. A positive detection requires the event to be an outlier with respect to the other events. For that, there is a definition of normal behavior learned from training data. The drawbacks of anomaly-based detection are a high number of false positives because legitimate behavior is usually highly diverse and dynamic and that the anomalies cannot name the actual root cause. An example for an anomaly is an unusually high data volume in the network when all the data assets are stolen and uploaded to the Internet. A third mechanism is the *policy*-based detection. A policy is a mix of both signatures and anomalies. The basic idea is to specify what is or allowed or forbidden, respectively. In contrast to signatures, policies are more general to cover the attack nature more generically. In contrast to anomalies, the deviation from a expected behavior is described explicitly. One of the drawback is that specifying policies requires expert knowledge about the system to protect.

Another difference between detection mechanisms is the location where they are deployed. The scope of *host-based* intrusion detection is restricted to events that directly come from the host as the supervised system. Supervising the hosts themselves result in many different and detailed information about the hosts that can be used in the analysis. However, this requires to deploy the mechanism on every host. In contrast, *network-based* intrusion detection is restricted to events about the hosts' communication. While this central monitoring and analysis scales better than having it deployed on every host, it can only detect intrusions from a perspective outside of the hosts. The mix of both perspectives is denoted as *hybrid* intrusion detection and combines the benefits of host- and network-based intrusion detection.

2.2.2 Challenges in Intrusion Detection

Challenges particularly in intrusion detection exist regarding the *audit data*. Significant storage capacities are required for the high data volume that auditing often generates. The high volume renders also any manual analysis impossible, which is why the analysis should be automated as much as possible. Another challenge is that the audit data is located on the supervised system itself and either logs or analysis results must be transferred to a central log server or analysis server, respectively. During both monitoring and transferring, the audit data must not be modified by the attacker even when the supervised system gets compromised. Also the expressiveness of audit data is a challenge when choosing which information is relevant in the first place.

Challenges in intrusion detection regarding the *analysis* include a limited efficiency of the analysis and a high number of false positives. The limited efficiency stems from the central approach that most IDSes follow. So-called agents collect audit data but the analysis is conducted on a central evaluation unit. Since the agents only forward the collected data but perform the analysis not even partly, the evaluation unit is the performance bottleneck. Is a problem particularly for distributed attack variants and attacks with parallel actions. The high number of false positive detection results is a problem that amplifies with the high data volume from

auditing. Even small false positive rates lead to an unmanageable number of false positives in practice.

Some challenges regarding the high data volume, especially the large number of alerts, is addressed by *alert correlation*. Its goal is to reduce the alert volume and highlight the bigger picture of an attack. However, alert correlation itself introduces new challenges as well. Most fundamentally, alert correlation must identify which alerts are related in the first place. A definition of such relations must somehow correspond to the definition of an attack, which is, however, generally rather intuitive than formally described. Similarly, correlated alerts are supposed to enable an immediate understanding of the attack and must therefore summarize the correlated alerts from an attack perspective. Apart from the challenges of individual techniques for assembling alerts to a bigger picture, the interpretation of the attacks is challenged by missing alerts. In case of false negative alerts, the correlated alerts only form an incomplete picture of the attack, while the goal is to reconstruct the full attack story.

In addition to the challenges regarding processing data in the IDS, the term *IDS evasion* summarizes more of them. An attacker can bypass the detection by interacting with the supervised system in an unintended way. Hiding the existence of interaction this way is known as covert channel [Wen+15]. Apart from bypassing the detection, the attacker can evade the particular detection mechanism in place. Specifically for evading signature detection, an attacker can obfuscate or mutate the attack such that it is semantically still equal but not matching the respective signature anymore. Specifically for evading anomaly detection, an attacker can transform the malicious actions to make them imitate benign behavior. An example for this kind of IDS evasion is the mimicry attack [WS02].

Another type of challenges is concerned with *privacy* and data protection. Auditing not only logs malicious activity but legitimate activities as well. Thus, the log data contains user identifying information such as user identifiers, i.e., user names, and documents their behavior, i.e., at which times an employee has been working. The respective data is collected without granting the users to determine themselves which data is collected regarding their person. If not secured properly, unauthorized parties might be able to access and abuse this sensitive data.

2.3 Classification of Intrusion Detection

Intrusion detection is an active field of research with many different challenges to overcome. Thus, the collection of approaches seen in practice and research is extremely broad. Thus, the following classifies the major disciplines in intrusion detection.

The detection of malicious activities as alerts is performed on the basis of either hosts or the network, and alert correlation forms the alerts to a bigger picture (cf. Section 2.4). Figure 2.2 summarizes the classification of these three disciplines. These are described with examples in the following.

2.3.1 Network Intrusion Detection

In network intrusion detection, the detection is based on either signatures, policies, or anomalies. While the first two are straight forward to implement in an IDS, many different techniques exists to implement the anomaly detection. Thus, only the network anomaly detection is elaborated in

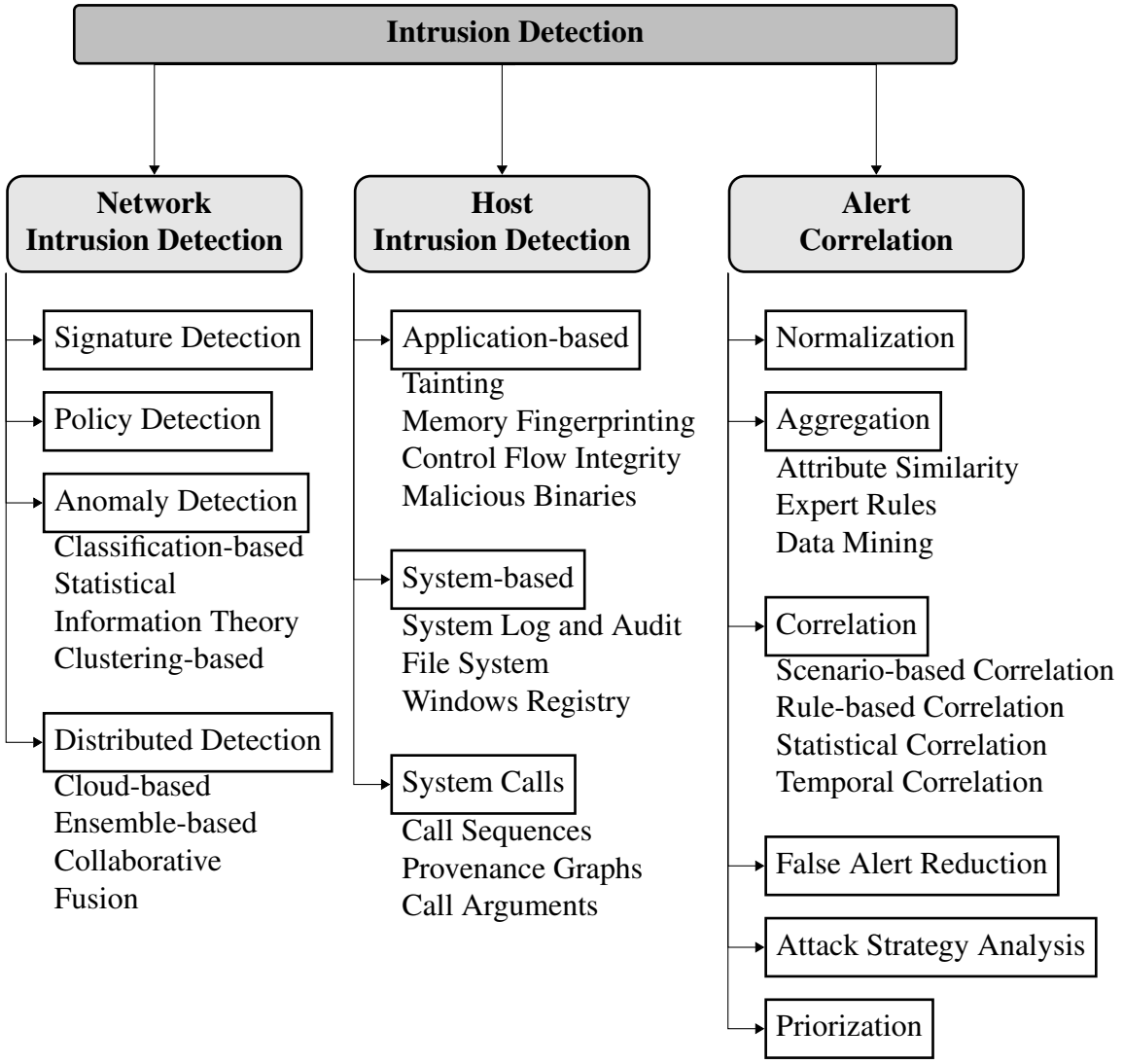


Figure 2.2: Classification of intrusion detection.

the following. Apart from the detection mechanism, a recent research trend is on distributed network detection.

Network Anomaly Detection Anomaly detection is a broad and active research field for the identification of network intrusions. The four major categories of anomaly detection techniques are listed here according to [AMH16].

- Classification-based Anomaly Detection:

  The classification-based approaches define a normal traffic activity profile with the help of event features that models the knowledge base about legitimate traffic. Any deviations from this baseline profile are considered anomalous. Feature selection or data reduction in general [Nis+18] is conducted before the actual classification is applied. The most popular detection techniques include support vector machine (SVM) [Esk+02], Bayesian network (BN) [Pea85; Kru+03a], Neural network and deep learning [Kwo+19], and rule-based [LSM99].

- Statistical Anomaly Detection:

  Approaches based on statistical theories create a profile of normal events and detects events that make the system in its entirety to deviate from the normal. Statistical techniques include mixture model [Esk00], signal processing [TJ03], and principal component analysis [Shy+03].

- Information Theory for Anomaly Detection:

  Approaches based on information theories [LX00] incorporate measures such as entropy, conditional entropy, relative entropy, information gain, and information cost to model the system's normal status and to detect deviations among the events.

- Clustering-based Anomaly Detection:

  The detection based on clusters leverages that legitimate events occur significantly more often. Also known as outlier detection, the events are clustered, and any events that are not close to a cluster's centroid are considered malicious. Clustering-based detection distinguishes between regular clustering and co-clustering [GN08].

Distributed Network Intrusion Detection Different variations of detection mechanisms form the basis for distributed intrusion detection regarding both distributed data sources and distributed classification of events.

- Cloud-based Network Intrusion Detection:

  Recently, research has began to investigate moving the detection mechanisms into the cloud [Kee+16]. To make this go along with performance benefits in terms of faster processing, the mechanisms must be suitable for a cloud computing environment. An example for this are detection mechanisms that scale well in such an environment when based on the MapReduce computing platform or the Hadoop Distributed File System (HDFS) [FMF14; Fra+11].

- Ensemble-based Network Intrusion Detection:

  Approaches that apply ensemble-based data mining [FS16] combine multiple classifiers and machine learning algorithms to achieve a better detection accuracy [Mir18]. Furthermore, the flexible structure of the approaches allow distributed implementations, which makes the handling of large and fast changing data efficient.

- Collaborative Network Intrusion Detection:

  In a collaborative intrusion detection system (CIDS) [Vas+15b], the sensors for collecting data are distributed in the network. In addition, each sensor also at least partly analyzes the data locally. The remaining data or intermediate analysis results, respectively, are exchanged with the other sensors to come to a joint detection result.

- Sensor and Information Fusion:

  Sensor fusion [ZKW15] incorporates data of different kind for intrusion detection. The heterogeneous data enables a classification that considers additional context about the events [AK17].

2.3.2 Host Intrusion Detection

The following paragraphs describe three major categories of host intrusion detection. They are derived from [Bri+19] with respect to the data on with they operate.

Application-based Host Intrusion Detection Often, a software bug in the application is the vulnerability that an attacker exploits to take over the system. Thus, detecting an unintended interaction with the application can indicate the intrusion early on. First, the logs written by the application itself about its operation state can be used for intrusion detection. The logs might directly include indicators for an attack like failed login attempts or indirectly indicate an attack in terms of warnings about a malfunction. Instead on relying on application logs, other approaches dissect the application in more detail.

A tainting framework [NS05] for running the application marks data from an untrusted source and tracks the processing of this data by the application. This way, code injection is detected when the untrusted input is about to be executed by the application as code. Abusing the application can also be detection by gathering the application's memory fingerprint and comparing it over time [VH17]. A deviation from the learned fingerprint indicates an intrusion. More explicit are detection approaches that check control flow integrity of the application [Gök+14]. A static analysis of the binary models the intended control flow reflected by the call sequence of code functions in the binary. A deviation from this model at runtime indicates a code reuse attack that is executing the program code in an unintended way. Other approaches try to detect malicious application before they even run, e.g., through the use of data-mining algorithms on the binaries [Sch+00].

System-based Host Intrusion Detection System-based approaches detect intrusions in the data produced by or on the level of the OS. The system logs in particular are a suitable data source for the detection of attacks related to privilege escalation. Thus, both User-to-Root (U2R) and Remote-to-Local (R2L) attacks can be detected based on the users' login behavior [Tch+15]. While system logs are coarse-grained, system audit data provides a chronological and more detailed description of the users' activity and networking information about connections. Events on this level are suited for intrusion detection with Markov chain [Ye+01] and SVM [LM04].

Also data about the file system is used for intrusion detection. File content and its integrity is one of the main concerns in this field [KS94]. Thus, research is developing efficient and scalable solutions to check for file integrity [Pat+04; KJL11]. Other approaches leverage the metadata and access patterns to perform intrusion detection [Gri+03].

Specifically for the Windows OS, the Windows registry is a data source for the detection of intrusions on OS level. This key-value database for the configuration of the OS, programs, and hardware is often modified by malware [HB06]. Such unauthorized modifications can be identified using anomaly detection [Hel+03].

System Calls for Host Intrusion Detection A more fine-grained data basis for intrusion detection on OS level is the monitoring of system calls. In contrast to system-based intrusion detection, the respective data is not provided by a system log but actively monitored by the intrusion detection in place [24]. Processes in the user space invoke system calls to interact with

the OS kernel for process, file, and network operations, among others. Prominent examples of system calls are the `read` and *write* operations. In general, a system call is invoked by a process running with the privileges of a particular user. Additional parameters might be passed depending on the invoked system call. Often, the parameters include an identifier to reference another system object such as processes, files, or sockets. For example, a process invokes the write system call passing a file descriptor (*fd*) and an array of bytes to make the kernel send the respective message over the particular network socket referenced by the given *fd* of the process. When invoking a system call, the kernel returns the call result, which can be a status code or a return value like received messages or object identifiers.

Approaches leverage the information of the monitored system calls in different ways to perform intrusion detection:

- Call Sequences:

  Various approaches are based on the sequences of system calls to profile the behavior of a program over a given time period. Most often, approaches apply variations of sequential features, i.e., n-Grams, [HFS98] or Markov models [GRS06] to classify a program as benign or malicious.

- System Object Dependency Graph:

  To model the interactions between different system objects, King et al. [KC03] introduces the so-called system object dependency graph (SODG). The edges between nodes in this graph, also called provenance graph, reflect the dependencies between caller and callee with respect to the monitored system calls. This knowledge is used to track the information flow across processes, files, and network sockets. It indirectly show which other objects a particular object is in control of or at least influencing by providing data input.

- Call Arguments:

  A third type of approaches leverages the system call parameters. Kruegel et al. [Kru+03b] analyze parameter features like string lengths and characters to build a models of normal arguments for each program. Deviations from this model indicate that the program is performing unusual kernel operations that result from the program being exploited.

2.3.3 Alert Correlation

Based on the alerts that are reported by an host- or network-based IDS, the correlation of alerts aims to reduce and fuse them for a meaningful view on the intrusion. This is especially necessary when having multiple potentially different IDSes in place. The six alert correlation components are listed here according to [SG06].

Normalization The IDS vendors have their own format for logging IDS alerts, which causes inter-operability issues when correlating the alerts. Thus, a common representation of alerts in terms of normalization is required. The most popular and mature approach for normalization is the Intrusion Detection Message Exchange Format (IDMEF) [DCF07]. This standard defines a data model for the representation of IDS alerts and how they can be exchanged over a network. Apart from describing the direct attributes of the alert, IDMEF allows the IDS to specify more

information about the detection such as the confidence in the validity of the analysis result. In additional, IDMEF supports the concept of alert correlation and can exchange correlation results by signaling the relations between previously sent alerts.

However, further problems in the normalization exists that are not solved by an alert exchange format such as IDMEF. An IDS still defines on its own the naming convention for certain descriptions. While some fields such as IP addresses and timestamps are well defined, others such as the classification to name the attack can be freely chosen by the alert provider. Resolving such inconsistencies among different IDSes is currently done by a mapping created from a database created with expert knowledge.

Aggregation The most effective component of alert correlation regarding volume reduction is the aggregation. In extension to finding equal alerts that just differ in their timestamps a little, alert clustering aims to find alerts with some equal attributes that have the same root cause or effect. Alert clustering distinguishes these three major classes:

- Attribute Similarity:

  Most similarity-based approaches assume that alerts from the same root cause will have similar attributes. Thus, similarity between alerts directly depends on the similarity between their attribute values. The similarity of attributes is usually a binary decision, values are either equal or different [VS01]. Few other approaches widen this definition of similarity by working with sets of already aggregated values [Jul03].

- Expert Rules:

  Other approaches define the similarity between alerts for clustering by expert rules. Still based on alert attributes, the work in [4] makes use of domain-specific expert rules based on predicate logic.

- Data Mining:

  Approached based on data mining [DC02] learn rules or mappings to decide which alerts belong together. The approach in [ZG06] uses a SVM to learn the statistical relations of the attack class for subsequent alerts based on the historical data. This can supersede the manual effort to define relations manually with expert knowledge.

Correlation The correlation component aims to find causal relations among the alerts. This component is an effective step towards the reconstruction of attack scenarios. Correlation distinguishes these four classes of approaches:

- Scenario-based Correlation:

  Approaches in this class correlate alerts based on the description of attack scenarios. Thus, the scenario must be known and modeled for detection. Languages such as LAMBDA [CO00] formally define the correlation and detection. For the detection, alerts are chained sequentially in various combinations according to the possible scenario definitions until an alert chain is found that fully matches the model [MD03]. A definition of attack scenarios can also be provided through machine learning on labeled data by identifying the scenario an alert belongs to [DC02].

- Rule-based Correlation:

  Rule-based correlation approaches define some type of pre- and postconditions of alerts and chain alerts whenever the postcondition matches the precondition of a subsequent alerts. This way, the chain of alerts evolves to attack scenarios that are not required to be known, as in JIGSAW [TL01]. Variations of matching rules allow partial satisfaction of conditions or define other restriction that have to be satisfied [CM02].

- Statistical Correlation:

  The causality between alerts can be derived from statistical observations. In [Cup01], a Bayesian network (BN) determines the conditional probability of an alerts to succeed in the presence of another particular alert. Based on this, alerts of the same scenario are identified.

- Temporal Correlation:

  To correlate two alerts based on their temporal relation, approaches apply some kind of time series analysis. The work in [QL03] makes use of the Granger causality test to determine if an alert X is forecasting another alert Y. If so, alert X is assumed to cause alert Y, which makes both alerts to belong to the same attack scenario.

False Alert Reduction The most effective component of alert correlation for the identification of false analysis results is the false alert reduction. This component aims to clarify whether a positive analysis result, i.e., an alert, is indeed a true positive or a false positive instead. Some approaches assume alerts that frequently occur together in a pattern to reflect true positives more likely [Man+00]. A technique to identify false positives is the use of additional context regarding the alerts. The network topology or alert semantics are only two examples of such context [Pie04]. Approaches use this context to verify if the attack can be successful in the first place, e.g., if the attempted exploit is applicable to the targeted system at all. Furthermore, the definition of attack scenarios can be used to identify false positives by considering alerts to be false positive if they cannot be correlated [YF04]. Another way to distinguish true positives and false positives is to assess the confidence in the analysis results of particular sensors and to fuse these confidences for correlated alerts from different sensors [YF05].

Attack Strategy Analysis The attack strategy analysis is looking at an attack's intention that the attacker pursues, mostly based on the already correlated alerts. The benefit of this analysis in comparison to the detection of particular attack scenarios is the tolerance regarding missing alerts, i.e., false negatives. This way, the attack strategy analysis is also used to predict upcoming attack steps that follow the previous steps. A method to model the series or combination of attack actions that achieve a particular attack goal is the usage of attack trees [QL04]. In this case, the root node represents the ultimate goal and sub-trees describe sub-goals that an attacker has to achieve first. Particularly for the hypothesizing and reasoning of missed alerts or correlated alerts, the work in [Nin+04] takes a causality correlation based on prerequisites and consequences and combines it with a correlation based on the similarity between alert attribute values.

Prioritization Some alerts are more important than others. Some indicate sever consequences in terms of violating security goals business needs, others require an immediate response. To assign priorities to alerts, various domain information are required such as security policies, network topologies, and vulnerability analysis, among others. The approach M-Correlator [PFV02] is an example that illustrates how to generate the required information, store it in a database, and leverage it to assign alerts an incident rank based on the outcome, priority and relevance of alerts.

2.4 Monitoring and Intrusion Detection Systems

There is a large market for intrusion detection solutions. Apart from traditional on-premise products, also more and more companies offer intrusion detection as a kind of service. Especially commercial solutions try to stand out from the crowd by extending detection capabilities for particular use-cases, combining different tasks related to intrusion detection to a all-in-one solution, or visualizing the security status on an abstraction level suitable for management boards.

However, all these products and services rely on very well-known principles and basics of intrusion detection (cf. Section 2.3). In their core, the security products can be narrowed down to a functionality similar to a traditional IDS, including the monitoring of the supervised system and conducting a security analysis of the events. With Zeek and osquery, the following describes a specific tool for security monitoring in the network and of hosts, respectively.

2.4.1 Zeek

Zeek is a network-based security tool for passive traffic analysis. This open-source security monitor inspects all traffic on a link in depth for various purposes. These purposes include analysis tasks even out the security domain like performance measuring or troubleshooting a network issue. Based on the capabilities that come with the wide range of analysis options, Zeek is also very suited for various security-related analysis tasks and detects signs of suspicious activity.

The time when Vern Paxson began the development in 1995, people were used to name this security monitor Bro. Recently in 2018, Vern Paxson announced the renaming to Zeek[1]. Back then in 1995, he was a computer science Ph.D. student at the University of California (UC), Berkeley, and a researcher at the Lawrence Berkeley National Laboratory (LBNL). The same lab already began to deploy Bro operationally a few years later as an early adopter. The first experiences with Bro have been presented at the USENIX Security Symposium in 1998 [Pax98] and later as a refined journal article in 1999 [Pax99]. After the young history of Bro, further development has been taken place at the International Computer Science Institute (ICSI) [2]. Long-term financial support has been coming from the National Science Foundation (NSF) [3] since 2003. That the initial creator of Zeek has been a researcher with academic background is visible throughout the whole history of Zeek including several related scientific publications at top-tier conferences. Thus, Zeek has been developed from the perspective of both researchers

1. https://zeek.org/2018/10/11/renaming-the-bro-project/
2. https://www.icsi.berkeley.edu/
3. https://www.nsf.gov/

and practitioners ever since. Nowadays, Zeek is used by largest supercomputing centers, national labs, university campuses, and even Fortune 10 companies.

Running Zeek in the network directly leads to a better understanding of what is going on in the network through the extensive log files written by Zeek. They achieve a high-level summary of network activities by describing every connection together with its statistics. In addition, more log files are written that contain application-specific information about web-based applications using the Hypertext Transfer Protocol (HTTP) protocol, the types of transmitted content in terms of Multipurpose Internet Mail Extensions (MIME) types, Domain Name System (DNS) queries and responses, encrypted traffic via TLS, email communication over the Simple Mail Transfer Protocol (SMTP), and much more. All these well-structured log files provide very valuable information about the network activity on an abstraction level that is suitable for the analysis with external analysis tools. Alternatively, these log files are further collected, processed, and visualized by SIEM systems.

All the valuable information and analysis result that Zeek writes to log files and beyond that is also processed live in Zeek itself for security-related analysis. For example, Zeek reports communication that is involving IP addresses or files known to be malicious, identifies software on the network that is known to be vulnerable in the installed version, reports TLS certificates that are untrusted or expired, detects SSH brute-force attacks as well as port scans, and much more.

The powerful functionality that comes with Zeek out of the box demonstrates only a fraction of what Zeek is capable of. Indeed, Zeek is a fully customizable and extensible platform for traffic analysis. The analysis is based on a Turing-complete scripting language with handy pre-built functionality for analysis in the network domain. Scripts written in this language implement the actual analysis tasks, including also a lot of the out of the box functionality such as logging. Thus, a Zeek deployment can be highly customized by enabling or disabling particular scripts. Furthermore, Zeek works with a package manager to share and deploy more scripts provided by the Zeek community. This flexibility is the major reason why some concepts of this thesis have been realized with Zeek.

With all these extensible and customizable functionality of Zeek, the developers themselves point out that Zeek is different to a classic signature-based IDS and was never meant so. Even though the possibility for signature detection exists, the actual power of Zeek comes with its scripting language that enables intrusion detection in various ways, including semantic misuse detection, anomaly detection, and behavioral analysis. From an architectural perspective, the security monitoring works in an event-based fashion and is layered into two major components: the core and the script interpreter. The event dispatcher in Zeek invokes functions in the core and handlers in the script land to process events of interest, that can emit new events on their own. For the internal transmission and queuing of events, Zeek leverages the publish-subscribe library Broker[4]. This library distributes events across the overlay of connected endpoints, even among different processes and remote machines. The incoming stream of network packets is first processed by the event engine, i.e., core, of Zeek. The events at this layer stand for network activity in policy-neutral terms, i.e., the decoded network packet stream in its different layers of the Internet model (cf. Section 2.1.1). Thus, these events just describe what is happening in the network. Thus, also application semantics like the requested Uniform Resource Identifier (URI) in an HTTP request are extracted from the packet stream in the Zeek core, and they themselves

4. https://docs.zeek.org/projects/broker

are reflected by events. The other component is the policy script interpreter that executes event handlers written in Zeek's scripting language. These scripts react upon specific events and perform further interpretation of them, e.g., to check if an observed Uniform Resource Locator (URL) is known to serve malware. These scripts write analysis or detection results to log files or send a notification directly to the security operations center (SOC).

2.4.2 osquery

The host sensor osquery [5] is an OS instrumentation framework for low-level OS analytics. osquery provides real-time insight into the current state of the infrastructure, i.e., end devices including servers, laptops, and desktop computers. Thus, osquery enables performance monitoring, compliance verification, and security checks of the whole network infrastructure.

osquery has initially been developed by Facebook with the intention to be deployed in-house for more insights into their infrastructure and released in 2014 as an open-source tool[6]. But also other large companies like Airbnb, Dropbox, and Netflix have seen the benefits of monitoring their infrastructure using osquery early on. While more and more interests and actors were involved into the development, the facebook open source project became a project with a diverse and active community soon. Recently in 2019, the Linux Foundation has taken over the mantel of osquery [7]. To address concerns regarding how Facebook handled the project in the past and that they could neglect the project in the future, the Linux Foundation announced the formation of a new foundation to support the osquery community. This foundation is dedicated to growing and sustaining a neutral osquery ecosystem.

The fundamental idea of osquery is to exposes the OS as a high-performance relational database. The SQL tables represent the current state of OS properties such as running processes, loaded kernel modules, open network connections, browser plugins, hardware events, file hashes, and much more. In the current version 4.4.0, in total 258 tables exists. Using SQL queries allows to explore these various OS tables, making use of regular SQL features such as joining different tables, filtering, or counting.

Analytics can make use of SQL queries in two operation modes. When running the interactive shell *osqueryi*, the query result reflects the current OS state, e.g., covering all running processes. osqueryi is the mode of choice when trying new queries or manually checking the status of particular hosts for forensic investigations. The automated mode is the daemon *osqueryd*, which regularly runs a schedule of queries. The daemon takes care of aggregating the query results over time and generating logs which indicate state changes in the infrastructure. Configuring osquery on all end devices in the network allows to schedule queries to be executed across the entire infrastructure. Apart from writing log files locally to the disk of the monitored hosts, remote logging capabilities as well as the possibility for integration into other log pipelines exists. For large deployments, osquery defines a representational state transfer (REST) application programming interface (API) over HTTP to a remote server to retrieve its configuration, including schedule, and to log query results. One project that implements this API on a remote server to manage an osquery fleet is doorman[8].

5. https://osquery.io/
6. https://engineering.fb.com/security/introducing-osquery/
7. https://www.linuxfoundation.org/press-release/2019/06/the-linux-foundation-announces-intent-to-form-new-foundation-to-support-osquery-community/
8. https://github.com/mwielgoszewski/doorman

In general, osquery is cross platform with support for Windows, MacOS, FreeBSD, and many Linux distributions. Thus, an SQL query is usually the same independent of the OS it is executed on. This makes monitoring of heterogeneous devices in the entire network easier. The running osquery itself makes sure that the query or OS table, respectively, is using the correct mechanisms for the particularly OS to retrieve the respective OS data. However, the full set of available tables is not applicable to every OS. While only some tables are supported on every OS (46 table in version 4.4.0), the remaining tables are only applicable to one or a few of the supported OSes.

From an architectural perspective, osquery is extensible through its modular plugins that add functionality regarding tables and logging. Such custom plugins are not even required to be compiled into the osquery binary. Instead, these plugins can be compiled in the form of an extension independently from the osquery binary. Such an extension runs as separate process next to osqueryi or osqueryd, respectively, and works without any modifications to the official osquery binary. The official osquery and the extension synchronize over a Thrift interface. Thus, the so-called osquery core running in the process of the official osquery, can make use of all tables and logging capabilities that are either compiled into the official binary of the extension. Table plugins define new osquery database tables and specify how the respective data is retrieved from the OS. Logging tables implement how the query results are persistently stored or forwarded.

2.5 Summary

This chapter has provided background information that is essential to understand and to motivate the detection approaches in the remainder of this thesis. An introduction to the fundamentals of computer networks has been given as wells as a summary of attacks that target the network at large. Such attacks threaten the network operation, critical business processes, and can have severe consequences to the overall business goals.

Furthermore, this chapter has summarized the basic principles of intrusion detection and highlighted the challenges that an IDS has to overcome. A classification of intrusion detection gives insights into the different mechanisms that approaches are based on. The most variety of mechanisms are found in the network-based intrusion detection and the anomaly detection in particular. Based on the alerts that come from both host- and network-based IDSes, also alert correlation has been shown to be very diverse in its goals and applied mechanisms.

The end of this chapter has presented the network-based security monitor Zeek as the main evaluation tool that is used throughout this thesis. In addition, the host sensor osquery has been presented as the counterpart to Zeek on hosts, which is also used in this thesis.

The next chapter discusses and analyzes the current state of the art in intrusion detection according to prior established requirements to an IDS.

3 Requirements and State of the Art

This chapter first describes requirements of an intrusion detection system (IDS) when performing intrusion detection. Afterwards, the intrusion detection process is introduced and utilized to give a classification of related work. The remaining sections then present and discuss state of the art along these requirements. Presenting state of the art starts with security information and event management (SIEM) systems that are widely used in practice for network-wide intrusion detection because they collect, correlate, and visualize information from various security-related logs. As most of the SIEM systems are commercial products, their applicability to solve the challenges in intrusion detection is discussed in general and separately from the remaining scientific research. Afterwards, the remainder of this chapter structures this remaining research work into three groups according to the intrusion detection process. The first group discusses approaches to detect network-wide attacks. The second group discusses approaches to correlate context to assist in intrusion detection. The third group discusses approaches to reconstruct multi-step attacks. In the end of this chapter, a discussion summarizes the requirements and the state of the art.

3.1 Requirements to Intrusion Detection

The following defines the requirements of large-scale intrusion detection in the network that are either functional or directly related to the quality of the system:

Detection Accuracy The IDS must achieve high accuracy in detecting attacks. The goal is to detect any attack and to report them as a single alarm. This also includes a precise and meaningful representation of the attack data belonging to an alarm. Any incorrect detection leads to undetected attacks (false negatives) or false alarms (false positives). High accuracy minimizes both.

Apart from the detection itself, i.e., the classification of malicious activities and their correlation to attacks, the accuracy depends on the quality of the monitoring data and its collection [Zho+18]. If monitoring, i.e., the capturing of activity, overlooks the malicious activity, it would not be evaluated and the attack not detected at all. However, as long as sufficient malicious activity is captured, an IDS must still be able to detect the attack.

Real-Time Detection The intrusion detection is referred to as *real-time* or *online* if malicious activity is identified right upon its occurrence. In particular, this means that the detection algorithm can report the attack as soon as sufficient evidence is observed. Real-time detection is required for initiating an effective attack response and mitigation already while the attack is still going on.

Efficiency The intrusion detection must perform efficiently and with low resource requirements. This means that the overhead for processing activities and alerts must be small. It must be possible to obtain a detection result with a reasonable amount of resources.

Scalability For the applicability in large networks, usually reflected by the number of hosts in the network, an IDS must be capable of handling an arbitrary large amount of data. To cover all monitored hosts and their traffic respectively, the intrusion detection performance must scale linearly with the amount of resource added [Hil90].

Easy Deployment The requirement is threefold and refers to the integration, configuration, and maintenance of an IDS and the environment that it is applied to. First, the monitoring and detection mechanisms need to be integrated into an existing environment, i.e., the network and the hosts, but should not require significant changes to them. Second, the deployment should have the ability to automatically adjust itself, without the intervention of an administrator. Last, the IDS itself should adjust to changing threats and new attacks.

Resilience and Self-protection An IDS must perform correctly all the time, even when the attacker is aware of the IDS in place and tries to sabotage it. The IDS itself might become a target first so that in the aftermath, the attacker can make the original attack go unnoticed. Thus, it is required that it cannot be deactivated by deliberate attacks.

More generally, an IDS must be resilient against internal failures and attacks. For this, it should still maintain its capability to report alarms with acceptable accuracy. For this reason, the detection should avoid any single points of failure (SPOF). Instead of a full IDS failure, internal failures and attacks should be countered with graceful degradation and fast restoration mechanisms.

Privacy As security-related data often contains sensitive information, exchanging such data needs to be protected, e.g., when shared in a distributed IDS. For this reason, the audience is carefully chosen to whom data is disclosed. Also privacy-enhancing technologies (PETs) like anonymization can help to achieve (collaborative) intrusion detection while preserving the privacy of involved users, companies, and network providers.

3.2 Classification according to the Intrusion Detection Process

The intrusion detection process guides through the different tasks that are required for successful intrusion detection, motivated by the alert correlation process in [HF19]. The intrusion detection process highlights the relations among monitored events, alerts, attacks, and multi-step attacks. After introducing the process and the tasks required to detect different kind of attacks, the process is utilized to define a classification of related work.

3.2.1 Intrusion Detection Process

During the intrusion detection process, security-related data is monitored and analyzed to detect any kind of attacks. For long time, intrusion detection research has been solely focusing on the classification of observed events and the issuing of alerts for every malicious event. However, recent research as shown that today's complex attacks usually comprise several actions and cannot be described nor detected by a single event. Therefore, their detection and reconstruction requires several detection and correlation steps. For that, the input data is processed and transformed to different representations that are explained next and illustrated in Figure 3.1.

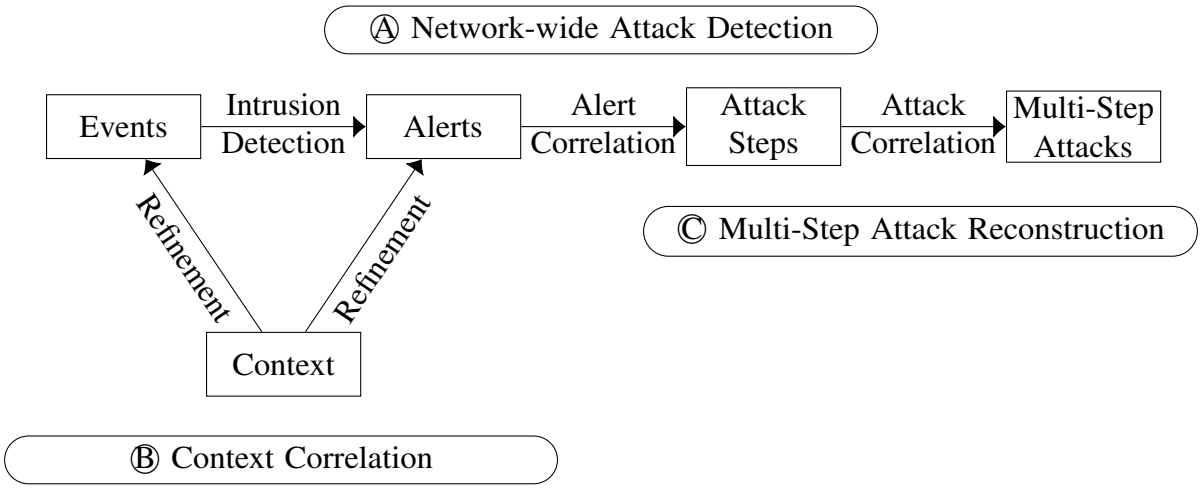


Figure 3.1: Intrusion detection process and classification of related work.

The input of the intrusion detection process are *events* that represent monitored activity, e.g., network packets, processes, and files. An event describes a low-level information in the monitored environment. Such an information includes the message bytes that are transmitted in the network, the execution of an application and its run-time parameters, or a file access along with its content. This information is represented as event features.

In the intrusion detection process, *intrusion detection* refers to the scope of a traditional IDS. By applying signature-, or anomaly, or policy-based detection, individual events are classified as malicious or benign. If malicious, an *alert* is generated containing the same features like the respective event and some detection specific features in addition to describe the type of alert. Example alerts are the packet payload of a worm or malicious mail attachment, the file of disk that is infected by a virus, or the failed authorization to an application.

To make results more accurate, *refinement* of the security-related data with additional *context* is necessary. Although context can be added at any stage of the intrusion detection process, the most value is usually added when used during event monitoring or intrusion detection. Often, the context comes from the correlation with other events or externally from a kind of knowledge database. For example, the context of a single event within the sequence of other events is highly relevant for intrusion detection, e.g., an incomplete Transmission Control Protocol (TCP) handshake as potentially part of a port scan can only be recognized when being aware of the three-way handshake and when setting the related messages into context. But there also more other examples for refinements with context. The alert of a detected worm message can be matched with the applications that actually run on the targeted host. This allows to prioritize alerts by determining the success probability of an attack. Furthermore, additional context can

be used in the intrusion detection itself, e.g., about IP addresses or domains that are known for malicious activities.

Based on monitored events and reported alerts, *alert correlation* is performed to assemble alerts that are related to the same *attack step*. For attacks that comprise a single activity, this solely relies on the intrusion detection, as the resulting alert already reflects the full attack. However, especially for attacks that comprise similar repeated actions, events and alerts are not isolated but must be seen in relation to reveal the full attack. This is the case for port scans or distributed denial-of-service (DDoS) attacks, where hundreds or thousands of connections belong to the same attack. There are actually two different approaches that are usually applied for the detection of such attacks. The first one is to rely on the intrusion detection and to actually correlate the alerts. This is often the case for attacks where all malicious events can be identified individually. In the other case, the malicious events can only be identified in combination. For that, the alert correlation might still operate on the basis of events but generates alerts that directly link the respective events. The outcome of the alert correlation are individual attack step. Each step eventually represents an unrelated attack but some steps might build upon each other.

The most complex attacks are multi-step attacks, i.e., such attacks where the attacker performs several different steps in chronological order such that the previous step enables the next step until the final goal is achieved. Compared to single-step attacks detected by alert correlation, *attack correlation* reveals the relation among the attack steps in a *multi-step attack*. The alerts of a multi-step attack are supposed to encompass all the different activities caused by the attacker across the multiple attack steps. As these steps build on each other, it allows to highlight the way and all the exploited vulnerabilities that lead to the successful attack.

3.2.2 Classification of Related Work

The intrusion detection process identifies the four different tasks *intrusion detection*, *refinement*, *alert correlation*, and *attack correlation* (cf. Section 3.2.1). However, these tasks not only operate on the output of the directly preceding task but can leverage all previous data and results from earlier stages or prepare for subsequent tasks. Therefore, intrusion detection approaches usually focus on a concrete task in the intrusion detection process but might also partly cover aspects of the other tasks.

For a successful intrusion detection in the network, individual solutions to these four tasks might exist but their interoperability must be given. Although being only one part of the intrusion detection process, the intrusion detection to assess events and to report malicious events as alerts is still the core of intrusion detection. That is probably why it has been extensively studied in the literature for decades [AMH16; Kwo+19; WS04; KT03; DD00]. However, detecting intrusions in large network, especially for distributed or multi-step attacks, cause other challenges. These cannot be solved by approaches that perform on individual events only. Thus, for the scope of this thesis, relevant work in intrusion detection is summarized in three classes and marked in Figure 3.1 with Ⓐ, Ⓑ, and Ⓒ. Figure 3.2 and the following paragraphs detail this classification that is also used to structure related work in Sections 3.4 - 3.6.

Network-wide Attack Detection The first class of related work contains detection approaches that are partly comparable to the scope of a traditional IDS. This group encompasses basic approaches that can be used in the first phase of the intrusion detection process to assess events

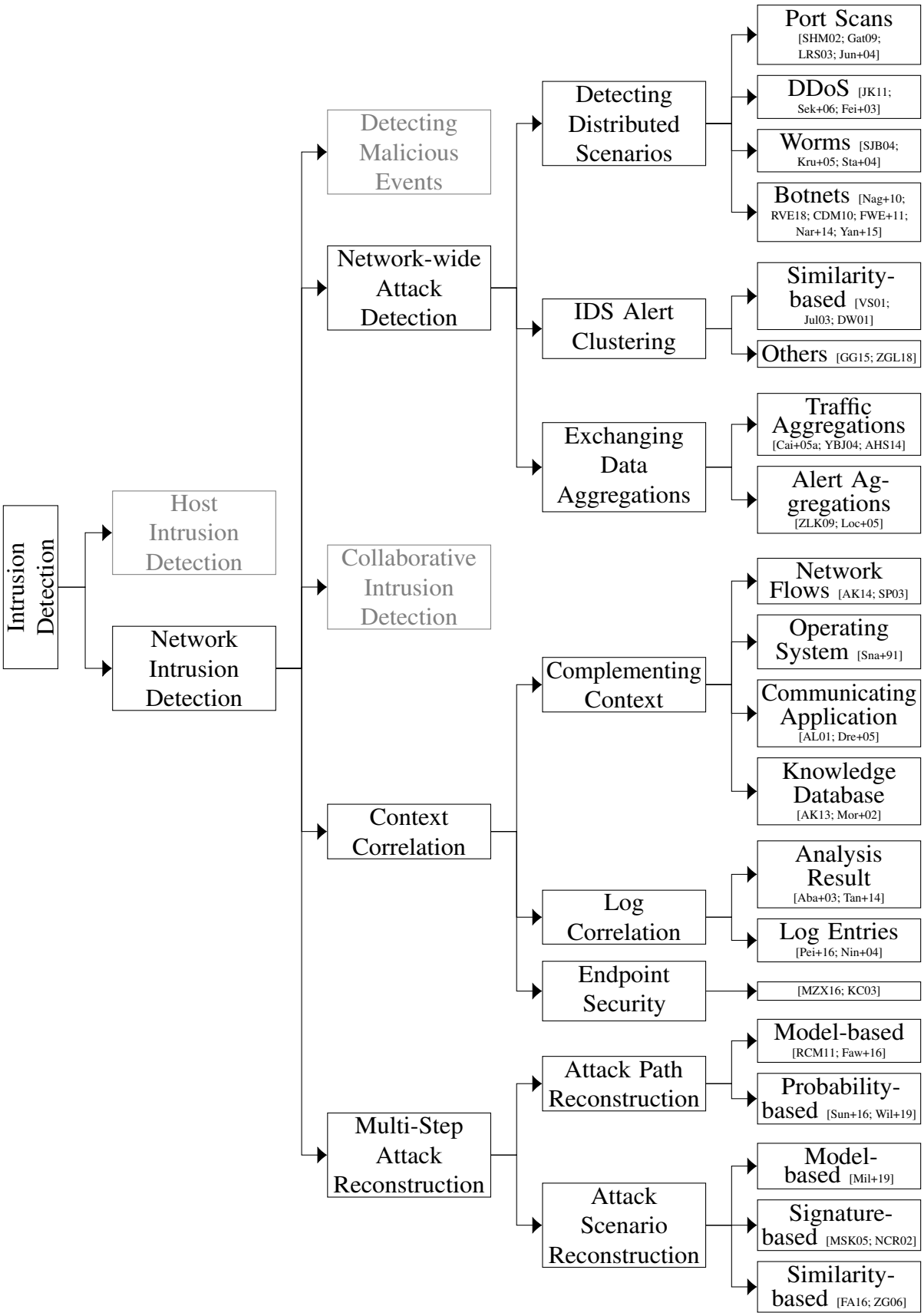


Figure 3.2: Classification of related work for intrusion detection. Grey classifications are out of scope for this thesis.

regarding their maliciousness, but also to consider relations among events and alerts to assemble them to attacks. The presentation of related work in this class focuses on approaches to detect well-known and network-wide attack scenarios that affects several hosts simultaneously or even larger parts of the network. As such distributed and large-scale attacks usually leave many traces at affected hosts and cause many alerts, clustering is a common technique used by the respective approaches. Approaches are further structured into three subclasses.

The first subclass contains approaches to detect specific *distributed attack scenarios*, usually on the basis of events. This subclass addresses attacks like:

- Port scans [SHM02; Gat09; LRS03; Jun+04]
- DDoS [JK11; Sek+06; Fei+03]
- Worms [SJB04; Kru+05; Sta+04]
- Botnets [Nag+10; RVE18; CDM10; FWE+11; Nar+14; Yan+15]

The second subclass contains alert correlation approaches that aim to reduce the alert volume to attack presentations. These approaches apply some kind of *clustering* or aggregation of alerts and their features. The result is a summary of alerts that are likely to belong together, eventually reflecting an attack that caused many similar alerts. For this reason, most of the approaches for this purpose are

- similarity-based [VS01; Jul03; DW01]. Others are based on
- entropy [GG15] or rough set theory [ZGL18].

The third subclass contains approaches that build a collaborative intrusion detection system (CIDS) [Vas+15b], particularly those that incorporate the *exchange of data aggregations* among sensor nodes. Such approaches also eventually reveal and report large-scale attacks, as a CIDS aims to detect attacks that are potentially relevant on a global scope. For data sharing, different languages and protocols exist, e.g., Structured Threat Information Expression (STIX) [Bar14], Trusted Automated Exchange of Intelligence Information (TAXII) [CDS14], or the Intrusion Detection Message Exchange Format (IDMEF) [DCF07]. However, the more interesting question is about which data actually has to be shared and how an algorithm uses it for intrusion detection. For that, CIDSes exist to share either

- traffic aggregations [Cai+05a; YBJ04; AHS14] or
- alert aggregations [ZLK09; Loc+05].

Context Correlation for Network Intrusion Detection The second class of related work contains approaches that assist in intrusion detection by providing additional context for events. Such approaches supplement security-related raw data and clustered data with additional contextual information. This includes related information to network connections, hosts, or alerts. Thus, the goal of approaches in the second group is to enrich the information value by correlating data from different domains. This strengthens the attack detection itself, i.e., filters false positives, or prepares for the multi-step detection in the next step. The approaches are further structured into three subclasses as follows.

The first subclass contains approaches to *complement context* of network events. The contextual information comes from one of the four sources,

- from the network flows themselves [AK14; SP03],
- from the Operating System [Sna+91],
- from the communicating application [AL01; Dre+05], or
- from a knowledge database about the hosts and their vulnerabilities [AK13; Mor+02].

The second subclass contains approaches that *correlate logs* of different semantics, effectively linking network logs with logs of the host domain. When linked to an alert, the correlated logs gives more insight into the attack. Depending on the log granularity, the correlation enriches log data on one of the following levels:

- Level of analysis logs [Aba+03; Tan+14], i.e. detection results
- Level of event logs [Pei+16; Nin+04], i.e., domain-specific activities

The last subclass contains approaches for *endpoint monitoring* that can provide additional host information for network intrusion detection. Approaches like [MZX16; KC03] neither perform intrusion nor are they directly incorporated into intrusion detection. But their outcome can assist in manual security investigations like forensics and threat-hunting.

Multi-Step Attack Reconstruction The third class of related work contains approaches that leverage all previous data from the intrusion detection process to perform a multi-step attack detection. This is necessary for all attacks that are composed out of several steps. Detecting these multi-step attacks requires to link the individual steps to reveal how the attack goal was achieved. The approaches are further structured into two subclasses.

The first subclass contains approaches for lateral movement. These approaches focus on the reconstruction of the *attack path* through infected hosts. There are

- model-based approaches [RCM11; Faw+16] to detect particular paths, and
- probability-based approach [Sun+16; Wil+19] to determine the most likely paths.

The second subclass contains approaches for reconstructing the *attack scenario*. These approaches focus on revealing the different actions performed during an multi-step attack. There are

- model-based approaches [Mil+19] to detect a particular attacker model,
- signature-based approaches [MSK05; NCR02] to link known intersections of steps, and
- similarity-based approaches [FA16; ZG06] to link equal alert features among steps.

3.3 SIEM Systems

In computer and enterprise networks, a lot of security-related data is usually logged. Based on these logs, security operators have to evaluate current threats and attacks to prioritize and initiate effective countermeasures. However, this decision making is usually difficult in practice because the log files are distributed in the network and contain information of different domains or granularity. Finding related log information across these log files is therefore difficult because of different semantics of the log entries. Alternatively, an isolated evaluation of single log files is neither sufficient for a sound assessment of the threat level in the whole network nor to detail the full story of a successful attack.

In fact, this requires data from several sources at different locations in the network. First of all, traditional IDSes – both in the network and on hosts – log their detection and analysis results. But their detection scope is limited to the monitored environment and usually excludes other data sources that are of high relevance for intrusion detection. In addition to malware detection through file analysis on the hosts by a host intrusion detection system (HIDS), their system logs include valuable information such as login events, privilege escalations, or changes to local security policies. But also unexpected high utilization of computer resources such as CPU or RAM can be an indication for misuse. Furthermore, especially server applications such as mail or web servers write valuable logs that summarize their operations and interactions with the network. In addition to packet analysis results by a network intrusion detection system (NIDS), more network functions such as routers or firewalls with their management and status information exist. And also other central systems of the network infrastructure such as the domain controller (DC) or the file share server can provide more insight into a security issue or enable its detection in the first place.

As all these logs from hosts and network functions are distributed in the network, a SIEM system [WN05; BMZ14] aims for their central collection, analysis, and visualization. SIEM systems are popular in many enterprise environments as they effectively enable security operators to identify many basic and frequent security issues in real-time. For that, it is explained next, how SIEM systems achieve to

1. collect the distributed data first,
2. then analyze it for intrusion or faults, and
3. in the end, visualize the analyzed data.

For the collection of distributed data, SIEM systems require a collector to run on the monitored hosts. It forwards the data of particular log files to a central storage for several reasons. First of all, this storage saves the full details of all retrieved logs in a tamper-resistant and audit-proof manner. The historical raw data is stored in a normalized and structured way so that it is suited for compliance purposes, forensics to investigate a security issue, and accounting reports. Apart from the benefits that come with the long-term storage capacities, the retrieved logs are also leveraged for real-time analysis.

The main purpose of the log analysis is to highlight security-relevant relations among the log entries through data correlation. SIEM products differ in the technology that they use for this purpose, including logical rules, machine learning, and artificial intelligence. The most basic analysis is data aggregation to calculate statistics and to detect when thresholds are exceeded. Prominent examples are the utilization of resources such as CPU and bandwidth but also

authentication attempts for Secure Shell (SSH) and web access, among others. Furthermore, events might be correlated based on timestamps to discover related events that are caused by the same root cause. For example, an unexpected high disk usage on a server and at the same time an increased network usage at the respective switch port of the server can be the result of data leakage. More sophisticated, SIEM systems can apply pattern detection to identify events that often occur together or in sequence. This enables security operators to identify patterns of regular operation and filter for anomalous events.

After collecting and correlating, SIEM systems report the data to security operators in an intelligible representation so that the important information and characteristics of an incident are clear. The most intuitive way is the visualization of the data in a dashboard. It usually highlights the current threat level and concrete incidents. Furthermore, it keeps track of historical measures to visually identify any trends that have to be mitigated early on. Furthermore, SIEM systems provide mechanisms for altering the security operator of critical events and threatening trends. This is usually done via well-established communication channels such as email.

Although SIEM systems seem to solve some of the challenges of intrusion detection, especially related to linking information from several sources, they also have drawbacks and limitations. The analysis quality is dependent on the analysis capabilities and the analysis language. Both, however, are only basic compared to state of the art technologies and approaches intrusion detection can make use of. The analysis, therefore, is usually limited to aggregations and pattern detection based on timestamps. Especially the latter, as well as other timestamp-based correlations of events, require precise and synchronized clocks across all monitored hosts. Any inconsistency of timestamps across logs can result in false positive and false negative correlation results. Another significant limitation of SIEM systems are the data sources. One drawback is the logs itself, as they are consumed unidirectional without any possibilities to dynamically control which data is retrieved for storage and analysis. Another drawback is the granularity of data, especially for actual network traffic data. Although some products include traffic statistics or even NetFlow [KB96] data, the analysis cannot make use of individual flow information because of the analysis limitations. However, accurate intrusion detection requires fine-grained network data such as full details of network traffic including packet headers and payload.

In summary, the main benefit of SIEM systems is the intelligible presentation of security-related data through their basic analysis, summary, and visualization. However, these systems lack fine-grained data and have only limited capabilities for analysis. Therefore, SIEM systems cannot take over all aspects of intrusion detection, especially the detection of sophisticated attacks. Instead, SIEM systems should be limited to consume only direct relevant and high-quality data that is produced by specialized systems for monitoring and intrusion detection. Consequently, there must exist other tools and systems that implement effective algorithms and mechanisms for intrusion detection that feed their results into SIEM systems.

3.4 Network-wide Attack Detection

This section presents approaches for the detection of network-wide attack scenarios, i.e., those that involve multiple attackers and/or multiple victims. Such attack scenarios like DDoS, scans, worms, or botnets, usually consist of several malicious network activities and thus cause a lot of similar data or alerts. The goal is to group the respective data to a single attack in an early stage of the intrusion detection process.

The approaches are structured into three groups (cf. Section 3.2.2). The first group contains approaches for the detection of specific distributed attack scenarios in network data. Approaches in the second group cluster alerts and those in the third group exchange data aggregations in a CIDS. Especially approaches in the last two groups summarize security-related data in the network to highlight its characteristics, eventually summarizing the network-wide attack.

3.4.1 Detecting Distributed Scenarios

The following paragraphs discuss detection approaches for specific distributed attack scenarios, including port scans, DDoS, worm spreading, and botnets.

SPICE Staniford et al. [SHM02] propose their Stealthy Probing and Intrusion Correlation Engine (SPICE) for the detection of port scans. The authors note that stealthy attackers can easily evade detection when originating not more than n distinct probes within m seconds from a single source. Based on their previous work GrIDS [Sta+96] that was not capable of detecting stealthy port scans and only correlated scans from the same IP address, SPICE now addresses the challenge of slow and randomized scans. To correlate scan activity across long time periods, Staniford et al. introduce techniques to identify ongoing scans and to efficiently compare new activity with historical activities. The architecture is composed of the monitoring sensor Statistical Packet Anomaly Detection Engine (SPADE) and the actual correlator SPICE.

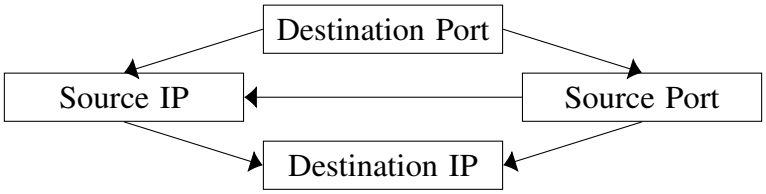


Figure 3.3: Dependable features used in the Bayesian network.

The anomaly sensor *SPADE* initially detects individual scan probes, i.e., packets to determine whether a single port is open or not. The detection of such probes is almost independent from the actual port status. Instead of focusing on failed connection attempts, the key idea is to detect connection attempts to IP and ports that are not expected to occur. For that, the likelihood for a particular connection with the well-known connection 5-tuple of IP addresses, ports, and protocol is predicted. The intuition behind this is that attackers want to extend their knowledge about a network and therefore will also probe ports and hosts for running services that are not known and accessed by the public. To detect such anomalies in service usage, a model with the probabilities for concrete connection 5-tuples must be trained for the network in question. Because counting every observed 5-tuples would be too inefficient, the authors propose to calculate the entropy of feature combinations as conditional probabilities and to model them in a Bayesian network (BN) [Pea85; Pea88]. Based on a real-world study, Staniford et al. suggest a BN structure as illustrated in Figure 3.3. The resulting conditional probabilities allow to determine the likelihood of any 5-tuples. Each conditional probability generally relies on only a few out of the five features. The combination with additional ones would add no notable value to the entropy. When monitoring the network with SPADE, it assigns an anomaly score to each packet. Packets that are sufficiently anomalous according to the trained model are passed to the correlation engine along with their anomaly scores.

The correlation engine *SPICE* identifies related scan probes, i.e., events, and groups them to port scans using a correlation graph $G = (V, E)$. Each node in V is an anonymous event and the edges E represent strong relations among the events. When adding a new event to the graph, the basic idea is to find the event with the strongest relation among all events observed so far. However, instead of calculating the strength between the new and all other events, SPICE makes use of simulated annealing [RN95]. For that, a previous event, i.e., $v \in V$, is initially chosen at random as current node. Starting there, G is traversed several hops along the existing edges E until a suitable node with high correlation strength is found. For every hop in the traversal, SPICE compares the correlation strength of the new events with both the current node and a random neighbor. If the correlation strength with this neighbor is higher, the traversal continues at this neighbor which becomes the current node. Alternatively, the traversal might continue probabilistically anyhow to converge towards a stronger relation multiple hops away. Otherwise, the traversal ends at the current node. In total, four traversal runs are performed that are likely to end in four different nodes because of the probabilistic nature. When adding the new node to G, edges to these four nodes are added as well.

The correlation strength in SPICE intuitively indicates scan activity that belongs to the same attacker. For that, the correlation considers characteristics of two categories:

- The *scan footprint* of a scan represents the set of destination IP address and port combinations an attacker is interested in.
- The *script* of a scan defines how it is performed, including the time sequence and the order of scanned IP addresses and ports.

Roughly speaking, the scan footprint and the script are equal for all scan probes of the same attack while they differ for probes of different attacks. However, many potential characteristics exist and their significance vary among the attacks. Therefore, Staniford et al. propose several heuristic functions $h_i(e_1, e_2)$ to test for specific characteristics among the features from the connection 5-tuples of two events e_1 and e_2. Weighting each function h_i with c_i, results in the correlation strength function f:

$$f(e_1, e_2) = \sum_{i=1}^{k} (c_i \times h_i(e_1, e_2))$$

The authors propose feature heuristics that test different feature combinations for equality, proximity, separation, or covariance. To report port scans in the correlation graph G, only edges with a correlation strength above a threshold remain. This ideally results in disconnected subgraphs, each grouping the events of a port scan. To further maintain G, outdated events must be removed. The lifetime of events is based on their anomaly score are calculated by SPADE. Events with higher anomalousness are saved longer. Also, events of a port scan group are only removed after the lifetime of every event in the group is over.

The outcome of SPICE are groups of scan events for individual port scans. The detection and correlation is performed in an online fashion and, therefore, applicable for real-time detection. Information about ongoing scans is saved longer so that future scan activity is assigned correctly to previous activity, even for stealthy scans that are slow and randomized. However, SPICE has been developed for scan attacks only and, therefore, is not universally applicable. Furthermore, the scan is just representation by the collection of all relevant events or alerts, respectively, with their full details, potentially including sensitive information.

Despite the overhead that usually comes with the comparison of every new events with a large amount of previous events, SPICE is efficient because of the simulated annealing. Also, the various parameters and heuristics of the correlation eventually allows the detector to adapt to high loads and to continue operation with decreased quality. However, this detection does not scale well because of the centralized processing.

SPICE can easily be deployed on top of a standard network monitoring solution that already provide basic monitoring of network flows.

Coordinated Scan Detection Gates [Gat09] proposes a detection algorithm for coordinated port scans. In such a coordinated attack, the attack tasks are distributed among the multiple sources under the control of the attacker. The author notes that such attacks usually indicate strong and powerful attackers that want to stay undetected. To assemble the scan activity from coordinated sources, Gates has developed an algorithm that leverages a fundamental characteristic of coordinate attacks. This is that coordinated sources distribute the load to achieve the common goal. Thus, the algorithm identifies scan activities that in combination covers a large portion of the network.

Attacker Characteristic	Symbol	Values
Number of ports	$\|P\|$	`One` port `Multiple` ports
Number of addresses	$\|A\|$	`One` address `Some` addresses `All` addresses of the network
Selection of addresses and ports	ς	In `random` order following `some` pattern the whole `subnet` or network
Selection of camouflage probes	κ	`None` Scanning of `random` IP addresses and ports Scanning to meet `some` property Scanning some contiguous space or `subnet`

Table 3.1: Characteristics to model the scan intention of an attacker.

A port scan is modeled based on the combination of four characteristics about the attacker. As shown in Table 3.1, this includes the ports, addresses, the sequence of scanned address and port, and the strategy to camouflage the scan, e.g., by sending additional probes from or to other IP addresses and ports. As the characteristics can have different values, in total 72 combinations of different characteristics exists, each representing an adversary class. However, Gates identifies only 21 of them to be sensible. Furthermore, the author defines the coverage and hit rate of a scan C to further characterize the scan activity from the defenders point of view, who does not know which might be the target and which might be camouflage:

- Coverage $\zeta(C)$: subnet of his network that is targeted by the adversary, i.e., the address range of the targeted IP addresses.
- Hit Rate $\mathscr{H}(C)$: is then defined as the percentage of target IP addresses within the scanned space.

Together, these characteristics define the footprint using the following tuple:

$$\mathscr{F} =< |P|, |A|, \zeta(C), \mathscr{H}(C), \varsigma, \kappa >$$

Based on this footprint $\mathscr{F}$ for individual scanners, Gates defines coordinated scans additionally by the number of sources $|S|$, the amount of overlap Φ in scan targets, and the algorithm $\mathscr{A}$ used to distribute the targets among the sources. When combining the footprints of the individual scans belonging a coordinated scan, it results in the overall footprint of the coordinated scan $< \mathscr{F}, |S|, \Phi, \mathscr{A} >$, where $\mathscr{F}$ characterizes the assembled scan activity of the coordinated scanners.

To detect the collusion between scanners, Gates is interested in scan footprints that fit together in such a way that some large portion of the entire space is covered. In contrast to other solutions for the set covering problem [Kar72], the amount of overlap between each of the scans should be minimized. Gates, therefore, proposes a modified version of the Altgreedy algorithm [GW97]. The modified algorithm tries to add in the smallest sets, i.e., scans, first, because for coordinated scanning, the larger scan is split into many smaller scans among the coordinated scanners. The algorithms loops over the different sets and assembles them such that the coverage is maximized while the overlap is minimized. For that, the author restricts the problem space to only the IP addresses and ports of the scan targets.

The result of the coordinated scan detection are horizontal and strobe scans against contiguous address spaces. This detection leverages an adversary model of coordination attacks and their characteristics, which is to distribute tasks without any redundancy, i.e., overlap among the tasks. This characteristic is incorporated into the solution to the set covering problem when assembling individual scans to coordinated scans that cover a larger target network.

The accuracy in the scope of coordinated port scans is good, but not applicable to detect attacks universally. However, the approach might be extended to at least any kind of coordinated attacks. The attacker model can be used as attack representation to characterize the attacker after detection. As the detection is triggered in fixed periodic time intervals, it effectively processes scan activity in batches and therefore does not perform in real-time.

As correlating the sets of scan events is performed by simple intersections of IP address sets, the detection is efficient. However, this centralized detection does not scale and implements no resilience measures against internal failures. But the attacker model incorporates camouflage such that the detection approach is able to compensate some IDS evasion.

Similar to SPICE [SHM02], the approach of coordinated scan detection [Gat09] can be easily deployed within passive network monitoring. But in contrast to SPICE, the scan footprint in coordinated scan detection can be cleared from internal IP addresses while still holding characteristics about the attacker.

Motif-based Anomaly Detection Juszczyszyn et al. [JK11] propose a characterization of network communication graphs for anomaly detection. The authors note, that the communication structure between hosts in a larger network changes in presence of a network-wide attack. The proposed method to measure deviations in the communication graph is based on the concept of network motifs [Mil+02].

Motifs are a measure originally known from the analysis of complex biological networks [BO04]. However, since then they have also been applied for data analysis in computer science, e.g.,

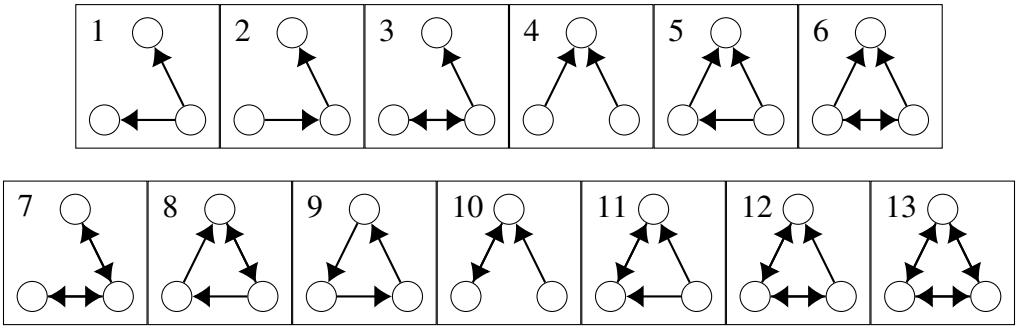


Figure 3.4: All possible 3-node-motifs with directed edges.

in the field of online social networks [Rot+17], application monitoring [ATF09], or network security [Har+16]. A motif is a particular small subgraph, usually consisting of three to seven nodes. Essential for a motif with a particular number of nodes is how they are interconnected with edges. In the case of subgraphs with three nodes, there exists 13 different motifs, i.e., patterns how these nodes can be interconnected with directed edges (cf. Figure 3.4). The key idea is that the counts of individual motifs characterize the whole graph in question. Furthermore, Milo et al. [Mil+02] define the so-called Z-score that indicates the statistical significance of each motif. The vector of all 13 Z-score values (in case of 3-node-motifs) characterizes the graph and is called Triad Significance Profile (TSP).

Juszczyszyn et al. evaluate their motif-based method for anomaly detection on communication graphs with two distributed attack scenarios. First, they simulate a network and benign traffic based on the Barabási-Albert model for scale-free networks [BA99] and embed worm scanning, i.e., a worm that spreads by scanning for new vulnerable hosts. Second, DDoS attacks are embedded in real-world traffic. For each time window during the experiments, Juszczyszyn et al. calculate and visualize the TSP of the communication graph. They conclude that the attacks in both experiments introduce notable deviations to the communication graph and the TSP, respectively. In particular, the presence of network-wide attacks will result in some motifs with high density to occur more often.

Although potentially applicable for the detection of distributed bulk attacks, the motif anomaly detection cannot identify individual attacks but can only indicate that something is malicious in the overall network communication. For that reason, the detection accuracy is low. Also, stealthy attacks will probably go unnoticed as their infrequent actions have no significant effect on the overall communication graph. As the detection result only indicates the presence of a potential attack without any details, it does not include any privacy-related information. Furthermore, because the detection assembles batches of NetFlows to communication graphs, it is not able to detect the anomaly in real-time.

As the anomaly detection triggers upon abnormal communication patterns among the hosts, it is resilient against attack variations as long as they still cause significant changes in the communication graph. However, the motif calculation is neither efficient because it is computational complex nor scalable because it cannot be distributed. As the detection is only based on coarse-grained communication patterns among the hosts, it is easy to deploy on top of passive network monitoring. However, it still requires the visibility of an Internet service provider (ISP) or larger to be effective.

BotGrep Nagaraja et al. [Nag+10] propose BotGrep, a detector for peer-to-peer (P2P) botnets (cf. Section 2.1.2). The authors note that the distributed and structured overlay topology of such botnets is of benefit for the botmaster, because of its resilience to churn. However, the characteristic communication among the bots in the overlay can also be used for detection. With BotGrep, the authors propose an algorithm that efficiently identifies P2P botnet communication in ISP networks and separates it from other legitimate communication.

The key insight for BotGrep is that the communication graph of P2P botnets has a structure with a fast *mixing time*, i.e., the convergence time of random walks to a stationary distribution. The goal of the detection algorithm, therefore, is to identify fast-mixing components in the communication graph $G = (V,E)$, where any kind of communication among the hots V is indicated by the edges E. The authors leverage short random walks on G to identify nodes in fast-mixing subgraphs, because their state probability mass is likely to be closer to the stationary distribution than nodes in slow-mixing subgraphs. For the calculation, BotGrep utilizes the vector q^t that saves the probability for each node i at any walk length t. It is initially set to $q_i^0 = 1/|V|$. This vector is recursively calculated as $q^t = q^{t-1}P$ using the transition matrix of the random walk, where d_i is the degree of node i:

$$P_{i,j} = \begin{cases} \frac{1}{d_i} & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

After an additional step to dampen negative effects on the probability values in q^t, the k-means clustering algorithm [Llo82] is used to extract similar nodes with similar probabilities. The result are candidate subgraphs $\{G_c \mid c \in 1 \ldots k\}$ with nodes and edges among them from G. Because an candidate subgraph G_c is likely include false positives as an artifact of the probabilistic nature inherent in random walks, BotGrep uses a modified version of the SybilInfer [DM09] framework to remove weakly connected nodes. In a last step, the authors highlight the need to recursively apply their modified SybilInfer clustering and propose several termination conditions.

In summary, the detection algorithm is content agnostic and therefore, it is not affected by the choice of ports, encryption, or other content-based stealth techniques that bots may use. Furthermore, the detection based on the communication graph is resilient out of two reasons. First, P2P botnet variations cannot hide their inherent distributed P2P communication as longs as the botnet activity is not too stealthy and infrequent. Second, the P2P nature probably stays visible in the graph even when sampling NetFlows under load. Although the detection cannot differentiate between P2P botnets and other P2P applications, once a P2P overlay is detected, the ground truth of a few participating nodes already allows classifying the remaining nodes in the respective overlay. Anyway, the output of BotGrep is a set of suspicious hosts that might run a P2P bot, which already is a good attack representation but still misses many other botnet-specific characteristics.

The detection based on the communication graph is not suitable for real-time detection because the monitored communication is processed in batches. But even for large communication graphs, the random walk approach with the transition matrix is efficient. Although the naive processing is centralized, BotGrep's analysis can be parallelized to be scalable. For the collaborative processes with different parties, privacy is preserved though a common identifier space. This way, BotGrep can be deployed even in networks smaller than those of an ISP when performing the detection collaboratively across multiple networks.

Deep Bot Learning Van Roosmalen et al. [RVE18] propose an approach for the detection of botnets to distinguish botnet from non-botnet traffic. For that, the authors apply deep learning on flows of TCP and User Datagram Protocol (UDP) packets. This has several advantages over other machine learning approaches when it comes to feature engineering and feature selection as well as efficiency [Ben+09]. Van Roosmalen et al. evaluate their approach via the detection of P2P botnets (cf. Section 2.1.2).

The main contribution of Van Roosmalen et al. is experimenting with different deep neural networks (DNNs) [Sze+17] and ladder networks [Val15] regarding their application in botnet detection. Compared to most other approaches that use some kind of higher-level features, e.g., NetFlow statistics, the authors rely on raw header information from packets. The DNNs are operated with supervised training or unsupervised pre-training using stacked denoising auto-encoders (SDAs) [Vin+08], which allow discovering higher-order features. A combination of unsupervised and supervised training is used as semi-supervised for the ladder networks. Based on their experiments with DNNs and ladder networks, Van Roosmalen et al. chose the most suited network configurations and hyper-parameters for the botnet detection.

For the evaluation, Van Roosmalen et al. highlight the need for mixed network traffic including data from P2P botnets. Mixing traffic from different sources should prevent the neural network to be trained towards recognizing the communication networks, rather than the traffic types. For their data sets with the botnets Storm [Hol+08], Waledac [Sto+09], and Zeus [And+13], the authors achieve a detection accuracy of 99.7%.

Although the approach accurately detects distribute P2P botnets, the detection requires to process individual network packets and, therefore, is probably not suited for many other distributed attacks that require to correlate several packets or events, respectively. But it is perfectly suited for stealthy botnet attacks and real-time detection as the time between malicious events is irrelevant and every event is assessed immediately. Trained on different data sets, this approach might be applicable to many other attacks.

The resilience against attack variations cannot be assessed, but it is assumed that the detection approach might still be able to detect unchanged or similar packets, even when most of the malicious packets significantly change and are not detected. Anyway, the whole detection is efficient, because of the applied deep learning method, but also does not scale for this reason. Furthermore, privacy was not a focus of this detection approach. It can be easily deployed in any environment with passive network monitoring. However, the approach must be trained for the respective attacks.

3.4.2 IDS Alert Clustering

The following paragraphs discuss alert clustering algorithms for the generic detection of network-wide attack scenarios.

Probabilistic Alert Correlation A seminal and early work on alert correlation was done by Valdes et al. [VS01] as an extension of their previous work [VS00]. The authors noted that their EMERALD [PN97] IDS sensors generate too many alerts to effectively process them manually. In addition, they noted that relations among attacks must also be visible in the relations among the respective alerts and among respective sensors. Because of this, they propose a concept for multi-sensor data fusion, which in some sense is similar to today's centralized CIDS.

Their work on Probabilistic Alert Correlation (PAC) [VS01] is actually more a clustering of alerts based on similarities among alert features, including source and target (hosts and ports), class, and timing information of the attack. The goal is to group similar alerts that together describe an individual attack. For that, PAC uses the notion of a meta-alert, that represents a collection of alerts by basically holding a list of values for each feature of the respective alerts. When receiving a new alert from the sensors, PAC merges it into the most similar existing meta-alert by updating the lists of feature values. If none of the existing meta-alerts is similar enough, the new alert leads to a new meta-alert.

PAC calculates the overall similarity between alerts and meta-alerts based on a pair-wise comparison of their features. Note that alert features usually have a single value whereas the features of meta-alerts are sets of values. Valdes et al. introduce different similarity metrics to compare both single values and set of values among each other. The average similarity among all features is between 0 and 1 and is weighted depending on expectations for specific features. For example, in a TCP SYN-Flood attack, the source IP is likely to be spoofed. Thus, expectations on this feature to match among the alerts can be relaxed.

Apart from the weighted overall similarity among all features, PAC introduces a situation-specific constraint that additionally must be satisfied to merge two alerts or meta-alerts, respectively. This constraint requires a minimum similarity for a specific feature, i.e., requires the values of this feature to be equal or at least very close. This minimum similarity is utilized to aggregate alerts to meta-alerts in three stages. At each stage, different features might be under constraint for minimum similarity and the features might be weighted differently. These three stages are:

1. **Thread**: Describes alerts of the same attack detected by the same sensor. Thus, high minimum similarity and high expectation is on the features sensor ID, attack class, source, and target.
2. **Incident**: Describes a specific attack with all alerts that might come from several heterogeneous sensors. Thus, minimum similarity and expectation are low on sensor ID, whereas expectation of attack class is moderately high and there is a minimum expectation on source and target.
3. **Report**: Describes the relation of various steps in a multistage attack. Thus, the minimum expectation on the attack class is relaxed compared to the incident stage.

When processing alerts according to these three stages, the outcome of PAC are alert clusters that represent individual attacks. The focus of this approach is on defining different correlation criteria for individual alert features. This is implemented by a flexible similarity comparison of alert features that can require equality of specific features and can balance the importance of individual features depending on specific situations.

PAC is universally applicable for alert correlation because of the multi-stage processing with flexible feature focus. There is also a focus that potentially detects distributed attacks. However, because of the batch processing of alerts, the correlation is not real-time capable and cannot detect stealthy attacks. Despite the accurate and flexible correlation to detect attack steps and basic relations among them, PAC represents attacks only by their full alert data but does not highlight its characteristics.

The correlation is complex $\mathcal{O}(n^2)$ regarding the number of alerts n and, therefore, inefficient as the correlation requires to calculate the similarity of new alerts to any other alert in the current batch. The multi-stage processing is organized hierarchical and ,thus, can be distributed to scale.

However, the depth of the hierarchy is tied to the three aggregations stages. This hierarchical processing is furthermore resilient against internal nodes failures by definition but not against overload situations.

PAC can be easily integrated into an IDS environment, as it only requires access to the alerts of deployed sensors. However, the correlation itself requires some expert knowledge about the attack scenarios to adjust the flexible feature focus at different processing stages. Also, this processing has no privacy measures in place.

Root Cause Analysis Julisch [Jul03] proposes a clustering of alerts for the analysis of root causes. The author noted that the high number of daily alerts distracts the analyst from identifying true positive and high-priority alerts. At the same time, irrelevant alerts are often a result of security-related issues, but are caused by other problems like network faults. Because of that, Julisch proposes to identify noisy root causes with redundant alerts and to eliminate them afterwards [JD02]. The experimental evaluation achieves a reduction of alerts by 87%. Based on the assumption that the same root cause triggers similar alerts, the proposed alert correlation technique efficiently identifies such large groups of redundant alerts. The correlation leverages a variant of the data mining technique called Attribute-oriented Induction (AOI) [HCC92; HCC93].

A detailed understanding of the alert data is required to apply the AOI technique. An alert consists of different features $a_1, \ldots, a_n$, including source and destination IP and ports, class, and time information of the attack. Each feature a_i has a range of potential values, denoted as domain $dom(a_i)$. For example, the value of the destination port is in $[0, 2^{16} - 1]$. Important for AOI are generalized attribute values that are not in $dom(a_i)$, but represent a subset of them. The relation among generalized and potential values is organized in a generalization hierarchy that is a connected directed acyclic graph (DAG) for each feature a_i. Leaves in this DAG represent one value out of the range of potential values $dom(a_i)$, whereas intermediate nodes reflect generalized values. Any intermediate node abstracts the values of its children, i.e., the leaves of the subtree that is rooted by the intermediate node. The root of the generalization hierarchy for feature a_i, therefore, represents the full $dom(a_i)$.

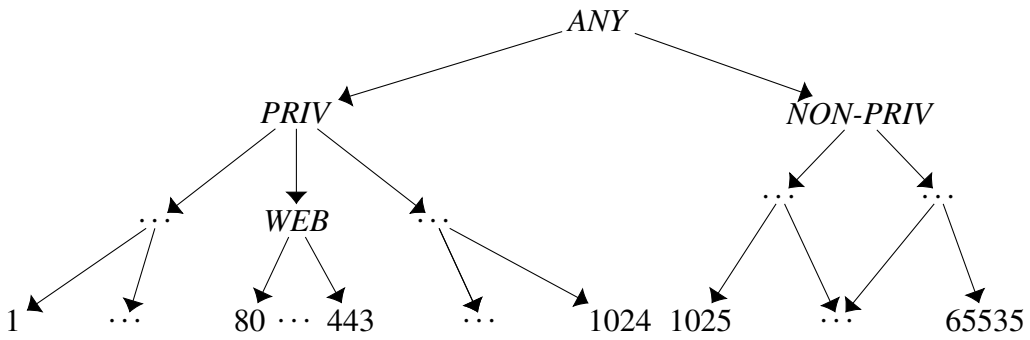


Figure 3.5: Example hierarchy for generalizing domain $dom(dst\_port)$.

The actual algorithm for alert clustering starts with the full set of alerts and repeatedly generalizes them. During their generalization, more and more of their feature values are overwritten by generalized values, consequently making the alerts more equal every iteration. Every time two alerts become fully equal across all their features, these duplicate alerts are merged and effectively clustered. The generalization goes on until an alert cluster exists that exceeds a minimum size.

The outcome is the generalized alert that represents all alerts in the respective cluster. Note that this algorithm terminates at the latest when all alert features are fully generalized. This then represents the full alert set. In particular, for each iteration of the algorithm, one feature is selected that is generalized by one level in the hierarchy for all alerts. Note that features can be generalized multiple times, e.g., destination port *80* can be for example first generalized to *web port*, then to *privileged port*, and finally to *any port* (cf. Figure 3.5). A heuristic is used to efficiently select the next feature for generalization that works as follows. For each feature, count how often the most frequent value exists. Generalize the feature with the lowest of these counts.

The result of this algorithm are clusters of similar alerts. In contrast to calculating the similarity among alerts based on a distance metric, this approach leverages generalization hierarchies. This way, the similarity between alerts is defined by equal generalizations of their feature values. The benefit of such a clustering are the respective generalized alerts, that not only represents the clustered alerts, but also describe them concisely.

The approach is most effective for root causes that cause many alerts, such as distributed attacks, and is thus potentially applicable for many attacks. But stealthy attacks will probably go unnoticed as their number of alerts per processing batch is probably not significant enough to be reported. The batch processing is also the reason why the correlation is not real-time capable. Anyway, when an attack is detected, the representation is a small data abstraction that highlights the attack characteristics by the most dominant alert features.

The correlation is efficient, especially because of the heuristic variant of AOI, but does not scale as the analysis cannot be parallelized. It is partly resilient against attack variations and obfuscation as the generalization in AOI filters any insignificant alerts and only reports alerts that follow a dominant pattern of equal features.

The root cause analysis is easy to deploy on top of an existing IDS. There is only low initial effort to define the generalization hierarchy for each feature domain that is, however, highly customized to the environment for some features such as IP addresses and ports. As the attack representation contains few details and abstracts most of them in the generalized alert form, the root cause analysis is partially privacy-preserving.

E-Correlator The E-Correlator by GhasemiGol et al. [GG15] is an alert correlation system based on entropy of alerts. The system's goal is to identify and cluster alerts that belong to the same attack. In contrast to others that rely on feature similarity among the alerts, E-Correlator defines the *alert partial entropy* (APE) to find alerts with the same quantity of information. The authors do this from raw alerts, without any predefined knowledge of attacks. Alert clusters are finally generalized to hyper-alerts and visualized as a hyper-alert graph (HG).

For their alert clustering, GhasemiGol et al. utilize the concept of entropy and adapt it to alerts. With the notion of entropy $H(X)$ of a discrete random variable X, the partial entropy $H_p(X = x_i)$ calculates the portion of the specific value $x_i \in X$ in $H(X)$. Adapted to alerts, APE for an alert is a vector of partial entropy for the individual alert features $F_j \in F$, including source and destination IP addresses and ports, protocol, and time. Intuitively, the partial entropy $H_p(F_j = f_{i,j})$ for a specific feature value $f_{i,j} \in F_j$ is the portion $f_{i,j}$ that contributes to the overall entropy $H(F_j)$ among all possible values of feature F_j. Furthermore, GhasemiGol et al. elaborate on the partial joint entropy to model dependent features, i.e., how to calculate the portion that the combination of values across several features contribute to the overall entropy. Anyway,

each alert is transferred to a vector of partial entropy values, which results in an APE matrix with rows representing individual alerts and columns holding the partial entropy values of their feature values.

Based on this APE matrix, the authors apply the density-based clustering algorithm density-based spatial clustering of applications with noise (DBSCAN) by transforming each alert with its APE as a k-dimensional vector. The result is a mapping of alerts to clusters that have a similar APE. The next step is to transfer each of these alert clusters into a hyper-alert that summarizes the feature values of contained alerts. Instead of creating simple lists of feature values as done by PAC [VS01] with meta-alerts, E-Correlator utilizes AOI, similar to how Julisch applied it for root cause analysis [Jul03]. The creation of hyper-alerts is based on the generalization of features. A generalization hierarchy defines a step-by-step mapping of feature values to higher-level correspondences [AZ09]. For example, IP addresses are generalized according to their role (Workstation, Firewall, Server,...) first and then to their network location (Intranet, demilitarized zone (DMZ), Internet,...). Generalization must be defined individually for each feature and custom needs. The root of the generalization hierarchies is always the *any* value. When defining and applying these generalization hierarchies to the alert clusters in E-Correlator, the result are hyper-alerts. Each hyper-alert feature is either a concrete value that is the same across all alerts or it is a generalized value that represents the values across all alerts.

The last step in E-Correlator is the reporting of the hyper-alerts. This last step includes an optional filtering to select only the most important hyper-alerts. Anyhow, the authors propose two metrics for ranking the selection of hyper-alerts. The first metric is the principle of maximum entropy, which basically means to rank hyper-alerts according to the accumulated partial entropy for contained alerts. The second metric is the hyper-alerts partial entropy that is the portion each hyper-alert contributes to the overall joint entropy among all possible hyper-alert value combinations. Finally, all or a selected subset of hyper-alerts are visualized for a high-level view. Nodes in the HG are the generalized or non-generalized IP addresses of a hyper-alert. The edges summarize several information of the hyper-alert, including number of alerts, protocols, source and destination ports.

In summary, E-Correlator combines techniques used in previous works but utilizes alert entropy instead of feature similarity among the alerts. In combination, this not only allows to cluster alerts, but the hyper-alerts also lead to a reduction ratio of the reported alarms of above 99%. Furthermore, because of the generalization, the hyper-alerts are an intelligible abstraction of the clustered alerts.

The flexibility and generality of E-Correlator based on entropy has some benefits and drawbacks. The benefits include potential universal applicability and resilience against attack variations because the entropy of alerts of the same attack might still stay equal. This holds especially for distributed attacks with many similar actions. But because of the alert processing in batches, the correlation probably fails for stealthy attacks and is also not real-time capable.

The attack representation in E-Correlator is valuable because of the generalized hyper-alerts and their visualization in the hyper-alert graph ranked according to their priority. The generalization of alert details also provides some privacy protection. The calculation of the APE matrix is efficient, because it can be easily calculated but does not scale. E-Correlator can be easily deployed on top of an existing IDS and does not require any export knowledge in operation.

3.4.3 Exchanging Data Aggregations

The following paragraphs discuss data structures that partly reflect an attacks and that are assembled to a global attack in a CIDS.

Worm Containment Cai et al. [Cai+05a] present a collection of services for the purpose of a *cyberspace defense system* [HCL05]. Based on their *NetShield* system [Hwa+05], they offer security services to be used as CIDS in a collaborative fashion. In particular, the authors focus on the mitigation of worms and attacks triggered by them, together denoted as worm containment. One service to achieve the containment is *WormShield* [Cai+05c] for fast detection of worm spreadings and another one is *DDoS Defense* [Cai+05b] for the traceback of DDoS attackers to worm infections. Following, both the system and the worm-related services are explained in detail.

The NetShield system is the fundamental communication platform used for worm containment, among others. The platform is built on a distributed hash table (DHT)-based overlay network (based on Chord [Sto+03]) for several purposes, including attack and intrusion monitoring, detection and defense, alert correlation, and security-update delivery across multiple administrative domains. It is supposed to operate on an Internet-wide scale. Within this system, every administrative domain performs security services locally and exchanges data with others to achieve a global view. The platform leverages the efficient routing, reliability, scalability, robustness to failure, and self-organization that is inherent to the DHT overlay. In particular, the organization of participants and content in the overlay into the same identifier space enables everyone to determine the identifier of the responsible participant that is most close to the identifier of the content in question. When sharing specific data that should be accumulated in the network, the DHT overlay deterministically determines the responsible participant for this task.

The first service towards worm containment is WormShield for the fast detection of worm spreadings on the Internet. The authors note that worms spread disperse, meaning that infections are initially dispersed over the entire Internet. At the same time, they note that worms spread by sending messages that are almost equal across infected systems. This allows to automatically derive worm signatures from the payload contents when infected systems widely scan the Internet to infect more systems. However, because of the initially dispersed spreading, a single network with its local visibility most likely overlooks the first traces of worm spreading as they are yet not significant enough among the total traffic of the network. Thus, WormShield utilizes the DHT overlay in NetShield to collectively accumulate traces of unknown worm spreading for fast and accurate detection, especially before most vulnerable hosts are infected. For that, each administrative domain in the NetShield system counts content blocks in packet payloads using the Rabin footprint algorithm [Rab81]. In addition to the count, they maintain a list of source and destination addresses associated with each content block. When the local counter exceeds a threshold, the information about this content block is shared in the overlay and accumulated. This way, the responsible node in the overlay assembles a global picture for the content block in question. It generates an alarm for worm spreadings if the accumulated count and the accumulated list of addresses both exceed a threshold on a global level.

Another service in the NetShield system regarding worms is the detection of DDoS flooding attacks that are often triggered by spreading worms [Cai+05b]. The problem is that the origin of the DDoS usually cannot be traced back. Deciding where the traffic physically came from,

i.e., the path it took, is broken down to a network with ingress and egress routers. DDoS traffic is entering the network from several ingress routers, i.e., the attack-transit routers, and leaves the network towards the target via a specific egress router. The challenge is to identify which of the ingress routers are actually attack-transit routers, based on the DDoS traffic observed at the egress router. Thus, the authors propose to utilize a traffic matrix $X_{i,j}$, which is a measure for identifying intersections of traffic S_i^+ that is entering at ingress router r_i and traffic S_j^- that is leaving at egress router r_j. However, collecting and correlating all traffic S^+ and S^- from all routers to calculate the matrix $X_{i,j}$ is practically infeasible. Thus, the authors propose a compression of S^+ and S^- instead of sharing the full traffic. In particular, they aggregate traffic in a key-pair-like fashion, i.e., counting how often a specific packet or flow is observed. Packets are identified by the invariant IP header fields and first few bytes of payload whereas flows are identified by the five-tuple identifier source and destination IP addresses and ports as well as the protocol. Sharing these identifier-based counters of each router allows estimating the traffic matrix $X_{i,j}$ and identifying where the DDoS traffic was entering the network from. Compared to sharing full traffic of N packets or flows, the required data volume to share is reduced to the order of $O(loglogN)$.

In summary, Cai et al. [Cai+05a] present collaborative detection methods for worm containment. Instead of sharing full traffic with others, they use traffic aggregations. They propose identifiers for packets, packet payloads, and flows for several purposes and share these identifiers along with their counters and summaries. This allows efficient identification of an Internet-wide worm spreading at an early stage and to efficiently traceback DDoS attacks that are triggered by spreading worms.

The accuracy of worm containment to detect global worm outbreaks is good under the reasonable assumption that the network payload of a worm is almost equal all the time. Even if a worm propagates slowly and stealthy, payload counters can remain and be updated in the DHT for long time until the counters exceed local and global thresholds. If the worm slows down too much to evade the distributed detection, the propagation would basically stall. With the aim to detect the distributed attack of a worm spreading, worm containment is not universally applicable. Also the detection is only partly performed in real-time as the counters at many local site have to exceed the threshold before they are merged on global level. Once a worm is detected, it is well represented because the payload associated with the exceeded counter characterizes the worm and can be leveraged for a detection signature. Furthermore, the representation goes along with some more spreading information like the IP addresses, which is helpful for attack mitigation but violates the privacy when shared with third parties.

The P2P nature of the overlay systems makes worm containment both scalable as the work is distributed among the P2P nodes and resilient against internal failures and overload situations as a local performance issue cannot cause global failures. But there is no solution to internal attackers or attack variations. The hashes of observed payload patterns can efficiently be calculated and shared in the DHT. The effort to deploy worm containment is low as it only requires passive network monitoring at several sites that collaborate in the detection.

Multi-Dimensional Alert Correlation Zhou et al. propose Multi-Dimensional Alert Correlation (MDAC) for collaborative intrusion detection [ZLK09] that works decentralized without a central correlation server. With multi-dimensional, they refer to a correlation that is not based on a single alert feature but multiple features at the same time. The authors noted that this

is required to capture the important characteristics of attacks. In their approach, they leverage multi-dimensional alert clustering to extract relevant patterns from alerts. Doing this in a decentralized-fashion in their CIDS leads to increased efficiency in terms of the time required to correlate alerts. Furthermore, the authors paid attention to the parameterization of their correlation algorithm such that a stealthy attack is detected on a global level, even when alerts are highly distributed among the individual IDSes.

The multi-dimensional alert correlation clusters alerts based on five main features of suspicious flows: source and destination IP addresses and ports as well as the protocol. For that, Zhou et al. propose an algorithm named *correlate-and-filter* that works in two steps, first correlating alerts to patterns and then filtering them. A pattern in this context is a combination of one to five of these features. The term pattern instance describes a combination of feature values that correspond to a specific pattern. In total, the authors identified eight patterns of interest $\{P_1 \dots P_8\}$ with combinations of one, two, three, or four features (cf. Figure 3.6). These correspond to a specific attack scenario:

- One-dimensional: most scans (P_1)
- Two-dimensional: flash crowds response [JKR02] (P_2), DDoS by Trinoo [CER99] (P_3), and most worm (P_4)
- Three-dimensional: reflector DoS [FBD15] (P_5), SYN Flood response [CER99] (P_6), and W32/Blast worm [Bai+05] (P_7)
- Four-dimensional: SQL-Slammer worm [Moo+03] (P_8)

These eight patterns, i.e., predefined combinations of features, effectively limit the search space for multi-dimensional alert patterns. Note, that the source IP is of extraordinary meaning to the authors and so this feature is included in each of the eight patterns. Based on the insight that these patterns build upon each other as illustrated in Figure 3.6, their relations among each other result in a lattice structure. This lattice starts with a single-feature pattern and becomes more specific to patterns with up to four features. In the decentralized two-stage correlation approach, each IDS participant first performs the correlate-and-filter algorithm on its local alerts and then the pre-processed patterns are assembled to a global view. The two steps of the algorithm are explained next.

The first correlation step maintains a so-called *pattern lattices* that hold all pattern instances observed in alerts for a specific source IP. Hence, each pattern lattice is rooted in a specific IP address. In contrast to the lattice in Figure 3.6, the pattern lattice holds several instances with different values combinations for the patterns P_2 - P_8 as respective values are observed in alerts. Pattern instances in the pattern lattice are linked according to the lattice in Figure 3.6 and according to common feature values. When performing the correlation step on local alerts, every alert increases the counter of the respective instance for each of the eight patterns. The second filter step aims to filter any insignificant or redundant pattern instance. Significance is defined locally by a support threshold δ_l that is a portion of the total number of alerts from all source IP addresses. Among the significant pattern instances that exceed this threshold, redundant patterns are eliminated. This happens to any pattern instance in the pattern lattice that is followed by another significant but more specific one.

The outcome of the correlate-and-filter algorithm are locally relevant pattern instances that are shared in the CIDS. Within the CIDS, the pre-processed pattern instances and their counts can be

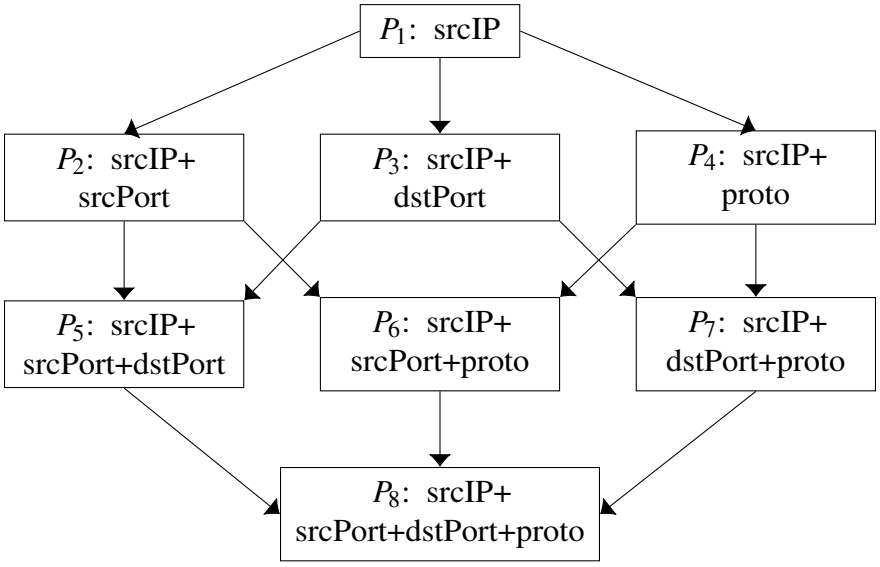


Figure 3.6: Relations among the eight patterns in the lattice structure.

accumulated to find the globally most relevant pattern instances. For that, the authors use a DHT-based P2P overlay with a publish-subscribe protocol. Each participant in this overlay consists of a detection unit that performs the correlate-and-filter algorithm locally and a correlation unit that is capable of retrieving pre-processed patterns and to accumulate them to global relevant pattern. P2P messages are indexed by the source address of the shared pattern so that pre-processed patterns can efficiently be routed to the alert correlation unit of the responsible participant that in return can notify others about globally relevant patterns for the respective source IP address.

One remaining problem in this two-stage correlation is how to set the local but also the global support threshold δ_l and δ_g, respectively when the correlate-and-filter algorithm is applied at both stages. The problem is as follows. Assuming a stealthy attack is detected in a centralized variant of the CIDS using a specific global threshold δ_g. When this attack is uniformly spread among the detection units, the question arises how low $\delta_l \leq \delta_g$ must be set such that every detection unit d_i with n_i alerts reports the respective pattern such that the accumulation of pre-processed patterns exceeds δ_g globally. For that, the authors propose a probabilistic model to estimate δ_l. For a given confidence, e.g., $\alpha = 0.01$, the local threshold δ_l is calculated as

$$\delta_l = \delta_g + Z_\alpha \sqrt{\frac{\delta_g(1-\delta_g)}{n_i}}$$

where the critical statistic is $P(Z \geq Z_\alpha) = 1 - \alpha$, e.g., $Z_{0.01} = -2.33$.

In summary, multi-dimensional alert correlation allows efficient correlation of alerts for malicious traffic from different network sites in the CIDS. It is efficient because the easy mining and counting of the patterns in the lattice structure. The outcome are intelligible abstractions of clustered alerts represented by their aggregated alert patterns, which highlights characteristic and common features among the alerts but misses scenario-specific characteristics. That the feature patterns are restricted by eight predefined distributed scenarios, results in only a moderate universal applicability. Because the alert correlation to identify these patterns is performed in batches, the detection is not real-time capable. The counters of the patterns are not updated continuously and might exceed the local and global thresholds only after the processing of the

next batch. But the probabilistic model for the thresholds enables to detect also stealthy attacks, even if not enough evidence is available in a single batch otherwise.

The fully distributed publish-subscribe overlay in the CIDS results in a scalable and resilient alert correlation. The deployment is easy as it requires only access to existing IDS alerts and requires no export knowledge apart from the predefined attack scenarios. When sharing the aggregated and characteristic alert patterns, third parties do not learn about the full alert data but still some IP addresses, which partly violates the privacy.

3.4.4 Summary of Network-wide Attack Detection

This section presented several approaches for the detection of network-wide attacks that have been discussed according to the requirements of an IDS provided at the beginning of this chapter. Not all requirements were explicitly investigated in the original papers. Thus, some discussion details are open to interpretation. Anyhow, Table 3.2 summarizes the overall results of this discussion. The detection approaches were classified according to their application field, which is either

- specific attack scenarios ([SHM02], [Gat09], [JK11], [Nag+10], [RVE18]),
- IDS alert clustering ([VS01], [Jul03], [GG15]) to find significant feature patterns among the alerts, or
- exchanging data Aggregations ([Cai+05a], [ZLK09]) to collaboratively assemble attack data of a larger scope.

The overall detection accuracy is assessed according to attacks in the intended scope of the approaches. Those for specific attack scenarios usually exploit the scenario characteristics and leverage them in the detection, e.g., the frequent failure of TCP handshakes or the communication behavior of P2P botnets. This usually leads to a high accuracy, except for [JK11] that cannot identify which events in particular are malicious.

In contrast, clustering approaches are usually based on some kind of similarity, e.g., feature or entropy similarity, which is not specific to a particular attack scenarios. While similarity is usually indeed caused among alerts of the same attack, this might also hold true for alerts of different but similar attacks, which results in the detection being biased towards one attack and to let the other one go unnoticed. To avoid treating different attacks falsely as one, additional attention has to be paid to individual attack instances as in [VS01]. If the similarity criteria are too narrow or too broad, the detection accuracy can still be okay but is likely to suffer.

The same reasoning applies to sharing approaches as they usually find locally significant and generic data patterns based on similarity. Even when [Cai+05a] focuses on the particular attack scenario of worm spreading, the detection is potentially error prone. It indicates a worm only by similar packet payloads but fails to incorporate the spreading characteristic, which is that after hosts are infected they start spreading themselves.

Also the more fine-grained aspects related to the detection accuracy allow to derive some common rules. As the selection of related work in this section focuses network-wide attacks, all approaches here detect distributed attacks, i.e., those that involves several attackers or victims. Although a port scan itself is in most cases already a distributed attack that targets many hosts in a large network, [SHM02] is likely to fail to link coordinated scanners that do not overlap

Network-wide Attack Detection										
	Attack Scenario					Clustering			Aggregations	
Requirements	*SPICE* [SHM02]	*Coordinated Scan Detection* [Gat09]	*Motif Anomaly* [JK11]	*BotGrep* [Nag+10]	*Deep Bot Learning* [RVE18]	*PAC* [VS01]	*Root Cause Analysis* [Jul03]	*E-Correlator* [GG15]	*Worm Containment* [Cai+05a]	*MDAC* [ZLK09]
Detection Accuracy (R1)	✓	✓	✗	✓	✓	✓	Ø	Ø	Ø	Ø
- Stealthy Attacks	✓	✗	✗	Ø	✓	✗	✗	✗	Ø	✓
- Distributed Attacks	Ø	✓	✓	✓	Ø	✓	✓	✓	✓	✓
- Attack Representation	✗	✓	✗	Ø	Ø	Ø	✓	✓	✓	Ø
- Universal Applicability	✗	✗	Ø	✗	Ø	✓	Ø	Ø	✗	Ø
Real-Time Detection (R2)	✓	✗	✗	✗	✓	✗	✗	✗	Ø	✗
Efficiency (R3)	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓
Scalability (R4)	✗	✗	✗	✓	✗	Ø	✗	✗	✓	✓
Easy Deployment (R5)	✓	✓	Ø	✓	✓	Ø	✓	✓	✓	✓
Resilience and Self-protection (R6)	✓	Ø	Ø	✓	Ø	Ø	Ø	Ø	✓	✓
Privacy (R7)	✗	✓	✓	✓	✗	✗	Ø	Ø	✗	Ø

Table 3.2: Summary of network-wide attack detection approaches regarding their compliance with the requirements given at the beginning of this chapter. Checkmark symbols ✓ indicate the fulfillment of these requirements, crossing symbols ✗ their non-fulfillment, average symbol Ø their partial match.

in their targets. In contrast, [Gat09] accounts also for colluding sources. There is also an issue with distributed attacks in [RVE18] when detecting P2P botnet traffic that is by definition a distributed attack but failing to link the traffic from several hosts to the same botnet.

Among the presented approaches some found a better way to represent the detect attacks than others. Most approaches at least label the related alerts with a scenario-specific label or in best case also characterize them to highlight the attack specifics. Only two approach completely fail this requirement. In [SHM02] the insignificant scans are not filtered but any alert ends up in an attack that is generically labeled as port scan without its characteristics. The anomaly detector [JK11] fails this requirement because it only indicates the presence of an network-wide attack without any details.

Only two approaches fully meet the real-time requirement. First, this is the botnet detector [Nag+10] that classifies individual traffic flows separately, It is therefore independent from the time between two malicious events but report the first one immediately. Second, the scan detector [SHM02] keeps the data structures of ongoing attack open and updates the currently

detected scans with every scan event. For the same two reasons both approaches detect also stealthy attacks, in addition to [ZLK09] where alerts can remain in the data structure potentially for a long time. Other approaches that process events or alerts in batches are neither real-time capable nor can they detect stealthy attacks. A special case is the approach [ZLK09] that partly covers the real-time requirement. The attack is not reported with the first local malicious event but immediately when enough evidence was seen at the global level.

Regarding the deployment effort, it is low or medium for all approaches. This is because there is only passive processing of security-related data, e.g., network sniffing for scenario-specific approach and access to IDS alerts for most clustering and sharing approaches. Both are easy to integrate into an existing environment. Additional deployment restrictions exist for [JK11] because this requires the network visibility of a large ISP or backbone network and for [VS01] because this requires extensive export knowledge about attack scenarios to model appropriate restriction on the alert features for correlation.

Apart from [JK11] and [VS01], all other approaches designed their detection algorithms to be efficient with low overhead. Scalability is only given by the approaches [Nag+10], [Cai+05a], and [ZLK09] that distribute their load among several machines, i.e., collaborating parties with their monitoring sites. As distributed attacks leave many traces, all approaches in this section have a basic resilience because they can inherently tolerate if a few events are not detected to be malicious or not mapped to the correct attack. Although not fully covering all other aspects of resilience, approaches are assessed better when additionally tolerating internal failures or attack variations.

The privacy is assessed best for those approaches that abstract the details of the malicious events when reported to humans or third parties. This ensures that no sensitive information is revealed. The attack abstraction in [Gat09] achieves this for example by an attacker footprint that is free from concrete IP addresses or ports. In the other cases as in [Nag+10], the identify of affected systems is hidden when shared with others for collaboration.

3.5 Context Correlation for Network Intrusion Detection

This section presents approaches for enriching security-related data and correlating context from different domains. It is used in the intrusion detection process to broaden the data scope and to refine the detection result by leveraging addition contextual data.

The approaches are structured into three groups (cf. Section 3.2.2). The first group contains approaches that extend the scope of network flows by correlating it with additional host data. Approaches in the second group correlate entries across different log files for intrusion detection. The last group contains approaches that provide correlated host activity that can be leveraged for network forensics.

3.5.1 Complementing Context for Network Activity

The following paragraphs discuss approaches that enrich network activities with complementing context for better network intrusion detection.

DIDS An early work on distributed intrusion detection that encompasses data from both hosts and the network is proposed by Snapp et al. [Sna+91] with their prototype Distributed Intrusion Detection System (DIDS). It performs distributed monitoring and data reduction utilizing monitors on individual hosts and in the network. The actual data analysis is done centrally by the so-called DIDS director that controls the monitors and retrieves their data. The focus of the authors is on the network-user identification problem that describes the tracking of a user moving across the network, possibly with a new user-id on each computer. In contrast to more recent research for the detection of lateral movement [Boh+17] or stepping stones [BGA19], DIDS assumes attackers that break into systems by solely making use of authorization methods, but not exploiting any software vulnerabilities to gain access.

To solve the problem of identifying the originating identity for a logged-in user on a host, Snapp et al. introduce the concept of a network-user identification (NID). The NID of a user that physically logs into a host is set to this local user account. However, if this original user now remotely logs into another machine, potentially under a different user, the remote login can be associated with the NID of the original user. This chain can be continued for any further remote login of the originating user. For the detection of such behavior, DIDS retrieves several notable host events as soon as they happen, including failed events, user authentications, changes to the security state of the system, and any network access such as *rlogin* and *rsh*. In general, the host events originate from audit trails as defined in class C2 – Controlled Access Protection – according to the Trusted Computer System Evaluation Criteria (TCSEC), frequently referred to as the Orange Book [Lip15]. For data reduction purposes, other than the notable host events are retrieved by the DIDS director in an aggregated fashion. The host events from the network monitor retrieved by the DIDS director include network activities such as *rlogin* and *telnet* connections, the use of security-related services, and changes in network traffic patterns.

Apart from using the host and network events to solve the network-user identification problem, the retrieved events can be analyzed at the DIDS director for several purposes. In general, the detection utilizes a rule-based expert system that transforms the raw audit events into high-level hypotheses about intrusions and about the overall security of the monitored environment. Table 3.3 describes the six levels of data processing for this purpose. Of particular interest for the network-user identification problem are the levels 1 to 3 that implement rules to track users by their NID. At upper levels, this allows us to aggregate the activities of the same original user over several hosts, even when using different user accounts locally on each host.

Level	Name	Explanation
6	Security State	Overall network security level
5	Threat	definition of categories of abuse
4	Context	event placed in context
3	Subject	definition and disambiguation of network user
2	Event	OS independent representation of user action
1	Data	Audit or OS provided data

Table 3.3: DIDS intrusion detection model.

The result is a correlation of network data with information about host activities. DIDS is demonstrated with the help of identifying network users to detect a single intruder that uses multiple accounts to launch an attack, such that the behavior can be recognized as suspicious only if one knows that it actually originates from a single source. The correlation itself is

universal applicable to set host activities into context and performs in real-time as activity in placed into context in an event-based fashion. However, there are no measures to make stealthy attacks visible. Also distributed attacks are not explicitly addressed but are easier to detect as shown with the help of the showcase. Furthermore, regarding the accuracy, DIDS models the intrusion data by aggregating it on six different levels towards representing the overall security state.

DIDS is highly privacy-invasive on lower levels of the DIDS intrusion detection model as it contains many details about user activities but it becomes more privacy-friendly on higher levels of the model when details are abstracted. Although the context comes from the hosts distributed in the network, the correlation itself is centralized and, therefore, not scalable. Neither is it resilient against attacks, failures, or overload situations.

Based on its description, the correlation might be efficient in the sense that no complex computations are performed. However, efficiency of the whole system is limited as high resources are generally required for the huge amount of host data. For the similar reasons, deploying DIDS is not easy because it requires the control and instruction of respective hosts to participate in intrusion detection by providing host context.

Transaction Inspection Almgren et al. [AL01] enhance intrusion detection by the concept of transaction inspection. The basic idea is to collect transaction information from a server application in real-time and to forward the data to a NIDS. This way, the intrusion detection can leverage not only the analysis of network packets but retrieves also host data specifically about the interpretation of the respective operation in the application. The result is an IDS that is coupled with the application itself instead of reading its logs.

When complementing traditional network-based and the new host-based data collection methods, it allows gaining a more detailed insight into transactions, e.g., to access the plaintext of an encrypted communication. For that, Almgren et al. propose to extend applications by a module that provides an interface to integrate a network monitor into the application. In particular, the authors use *eXpert-HTTP* from EMERALD [PN97] as analysis engine. For a proof of concept, the authors extend the web server Apache by a module that forwards data to eXpert-HTTP. The authors noted that the internal logging in Apache is a reasonable place to insert their module for forwarding relevant data. This is because there is a well-defined application programming interface (API) from which their module can receive structured data and because it allows the module to register to hooks and callbacks that are invoked in several stages when Apache processes a request.

When considering a particular Hypertext Transfer Protocol (HTTP) request to the Apache web server and the respective response to be a transaction, the analysis of a NIDS is limited to the bytes transmitted in the packets. With their integration into Apache, Almgren et al. can now leverage addition host context from the application at the several processing stages. This includes the Uniform Resource Identifier (URI) translation that determines the path and file to be accessed, the header parsing, access control, authentication, authorization, and Multipurpose Internet Mail Extensions (MIME) type checking.

The result is a correlation of application semantics with the packets seen by a NIDS. The closely tailoring to be part of the application leads to more information than external monitors could achieve, including the interpretation of an operation on application-level. As the NIDS now

has the ground truth about how a network packet is processed by the application, the detection accuracy of malicious network traffic can be increased.

Getting to the internals of an application usually requires to customize it, which makes the deployment hard or even impossible. But it is eventually universal applicable to any application. The transaction inspection is most effective for stealthy attacks that cover their actions in encrypted traffic but does not help in case of distributed attacks as this scope are individual transactions, i.e., requests and their responses. The inspected transactions are not represented in any intelligible way but they are efficiently inspected as this requires only to match ongoing network communication with the respective application internals. This happens in real-time because of the event-based fashion in which internals are received and correlated.

The application internals potentially reveal user sensitive information and, therefore, this approach violates the users' privacy. The whole correlation and process is centralized and does not scale. Also, there are no resilience measures in place against internal failures, attacks, or overload situations.

3.5.2 Intrusion Detection through Log Correlation

The following paragraphs discuss approaches that correlate information from different log files for better network intrusion detection.

Anomaly Correlation Abad et al. [Aba+03] present their work on anomaly correlation to increase the accuracy of anomaly detection across different log files. The authors assume that attacks leave attack traces in several log files. Thus, Abad et al. correlate the anomaly detection of the different log files to be more confident in their detection result. The anomaly detection of individual log files is based on the data mining software tool RIPPER [Coh95]. The used data mining techniques filter out the important log data and identify different attacks that may have occurred.

The anomaly detection itself works on the principle of predicting the next element in a sequence. As a proof of concept, Abad et al. implement this for the logs of system calls and network traffic. For the log of system calls, the authors define a sequence by subsequent operations such as *open* and *close*. For the log of network traffic, a single element in the sequence is defined by the number of network flows within consecutive 10 seconds. Predicting the next element in a sequence of system calls or network traffic is based on the last five to 19 elements. When RIPPER is trained with a legitimate data set, the outcome are prediction rules for the next element in a sequence. The resulting rules are then tested to associate each rule with a confidence level that reflects in how many test cases the rule was right in predicting the next elements in a sequence.

When applying the prediction rules to a continuous stream of elements, i.e., system calls or number of network flows, each observed element can either match the prediction or differ from it. When it differs, the element is assigned a penalty that equals the confidence of the violated prediction rule. Further processing of individual element streams is done utilizing a sliding window of length 13. To assign a new element in the stream the label *normal* or *abnormal*, not only its own but also the prediction of the last twelve elements are considered. Thus, for this decision, in total 13 elements and their predicted values are evaluated. For that, the penalties of any mismatches among these 13 elements are accumulated. When this accumulated penalty of

the sliding window is greater than half of the number of elements in it, i.e., greater than 6.5, the new element is classified as *abnormal*. In fact, this classification is not done individually for each log file but dependent on each other. To actually correlate the anomaly detection across system calls and network traffic, the threshold for classification on each log file depends on the current accumulated penalty of the other log file. Intuitively, the penalty of the one log file is incorporated into the classification of the other log file by lower its threshold. This way, there is more confidence in true positive and true negative classifications of *abnormal* log entries.

In summary, the anomaly correlation based on system calls and network traffic assumes abnormal activities to be present in both logs. However, the total required confidence can be balanced between the two logs. Less abnormal activity in one log requires more abnormal activity in the other log to detect an intrusion. Rephrasing this, already high abnormal activity in one log requires only low abnormal activity in the other log to detect an intrusion.

Apart from the initial learning phase of system call and traffic sequences with RIPPER, the prediction and its comparison with observed sequences is efficient. At operation, deviations from known sequences indicate an attack is going on, not tailored to stealthy and distributed attacks but potentially universally applicable. The detection result reports the deviation but cannot abstract the attack in any meaningful way. That an attack is taking place reveals no sensitive data, but the collection of system calls poses a high threat to privacy. As this hosts data is retrieved and processed in an event-based fashion, the detection is real-time capable.

The anomaly correlation is neither scalable nor resilient because of the centralized processing that cannot gracefully degrade its operation. Furthermore, compared to the easy collection of network traffic statistics, the retrieval of system calls from hosts requires more effort to deploy the detection.

HERCULE Pei et al. [Pei+16] propose HERCULE to reconstruct multi-step attacks via community discovery on a correlated log graph. The authors note that complex attacks encompass several stages with each stage involving a different activity on the host, e.g., the download of an Internet file or the start of a new process. The authors make HERCULE to observe such activities by leveraging lightweight logs on the hosts and to correlate their log entries to a log graph. From this graph, HERCULE extracts attack communities, i.e., log entries that are related to the same multi-step attack. This allows reconstructing the panoramic view of the complex attack.

HERCULE parses six log files related to resolving Domain Name System (DNS) names, establishing network connections, accessing HTTP websites, creating processes, accessing file objects, and attempting authentication. In total 20 log fields such as *timestamp*, *pid*, and *GET path*, are extracted from the log entries among the six log files. Intuitively, two log entries must have a common field value to be correlated. This can be a match on the same field, e.g., the *timestamp* that is available in all log files or the *process name* that is available for process creations but also for object access, among others. However, a match can also happen between different fields, e.g., the destination of a connection equals the resolved IP address in DNS. In total, the authors define 29 such combinations to match fields within and across log files. Based on the six log files, HERCULE creates a correlated log graph $G = (V, E, D)$ with the nodes $v_i \in V$ being individual log entries and edges $(v_1, v_2) \in E$ among them when at least one match among the 29 combinations exists for the field values of (v_1, v_2). In addition, each edge holds a vector of length 29 that encodes which of the combinations D matches.

Based on this correlated log graph, HERCULE extracts communities [MV13], i.e., clusters of log entries that show high density among each other with respect to equal log fields. From a formal perspective, community clustering aims to maximize the edges within a community and to minimize the edges between communities. However, the authors notice that not every one of the 29 combinations of log fields has the same strength, but an edge is added to the graph whenever at least one combination exists. For that, Pei et al. optimize the community detection by introducing weights to the edges. This weight is based on a weight function that takes the strength of each combination into account and depends on the combination vector, that is assigned to every edge. In particular, quadratic programming is chosen for this purpose as it outperforms other weight assignment algorithms that are compared by the authors. Finally, HERCULE efficiently extracts communities on the weighted graph using the Louvain method [Blo+08].

The outcome of HERCULE are clustered log entries that come from log files with diverse information, both host- and network-related. Related log entries abstract a sequence of related activities, potentially linked among hosts. If it is possible to match an alert to a log entry, then any cluster that contains this log entry is marked as an attack. Consequently, the contained log entries are supposed to reflect the sequential activities of the attacker.

HERCULE is not efficient as the graph creation requires to search for equal features potentially among all other log entries, which results in an complexity of $\mathcal{O}(|V|^2)$ where $|V|$ is the number of nodes, i.e., log entries. The later community forming is triggered manually for the batch of current log entries, which is the reason why HERCULE is not real-time capable. The whole analysis is neither scalable nor resilient. Also, it is not easy to deploy because this requires to retrieve several logs from different places in the network and the active participation of hosts.

HERCULE is universally applicable to any attack scenarios that leave traces in the respective log files. In contrast to stealthy attacks, distributed attacks that target several different hosts or services might be easier to detect with the help of HERCULE. The analysis outcome, i.e., groups of log entries, still contain all details including any sensitive user information, which neither abstracts the attacks with its characteristics nor does is protect the privacy of the users.

3.5.3 Endpoint Security for Network Forensics

The following paragraph discusses an approach that provides insights into host context for better network intrusion detection.

ProTracer Ma et al. [MZX16] present ProTracer, a lightweight system for provenance tracing. The term provenance describes the origin of information flow, e.g., the network connections of a process or the process that was writing to a memory location. The authors identified two principal mechanisms to achieve provenance tracing, but note that both have limited practicality. First, provenance tracking using system-level audit logs [Kin+05] (cf. Section 2.3.2) suffers from dependence explosion and therefore cannot be used in an online fashion. Second, also most provenance propagation or tainting [Mun+09] approaches have a high overhead because of heavy-weight instrumentation. With their ProTracer, Ma et al. combine the best of both provenance mechanisms while achieving their goals with low overhead. First, the *what*-provenance should answer on which objects another one depends on. And second, the *how*-provenance should

detail the dependency with other objects. The outcome allows querying any system objects for what and how both backward and forward.

The authors achieve their goals with a low overhead for several reasons. First, they implement a lightweight kernel module for system call monitoring that is based on so-called tracepoints instead of using the more inefficient Linux Audit system. Second, a user space daemon receives the monitored calls as events and processes them on the fly. This daemon, in fact, alternates between logging and tainting the provenance information in the events. It only logs when events conduct changes to the permanent storage or the external environment such as writing a file and sending a packet. For other events such as file reads and network receives, the daemon performs coarse-grained tainting of the object that might propagate with further events. Tainting instead of permanent logging effectively lowers the overall overhead because a filtering happens for two reasons. It avoids to log redundant events and avoids to log dead events, i.e., those that have no effects on provenance propagation.

To further optimize the performance of ProTracer, the authors leverage the concept of unit levels as proposed in their previous work on BEEP [LZX13]. It separates the runtime execution of a program into units, with each one representing independent tasks like processing a request. This reduces false positives, as information flows can be isolated through the whole lifetime of a process. Consequently, logging and tainting is based on process units and the other system objects files and sockets.

The resulting logs filter redundant or irrelevant provenance information while being precise and complete on provenance propagation. Therefore, this substantially reduces the space consumption and the size of provenance graphs that are generated based on these logs and allows us to efficiently find causally related activities while investigating a security incident.

The custom kernel module for the monitored hosts has some benefits and drawbacks. The main drawback is the run-time modification of Linux at the kernel level. But this enables several benefits regarding the retrieval and tainting of data. First, it is partly scalable because of leveraging threats. Second, it is efficient in the sense of less overhead compared to the Linux kernel audit, both regarding receiving kernel data and log volume because provenance information is filtered before logging. The tainting for this provenance information happens in an event-based fashion and, therefore, in real-time.

As provenance information is hold as long as it potentially propagates further, ProTracer is suited to assist in detecting stealthy but not in distributed attacks. Anyway, its applicability is universal as provenance tracing is independent from attack scenarios and characteristics. For this scope, ProTracer provides a good representation of the results by logging objects tainted with their provenance and abstracting the many related objects. This also reduces the amount of sensitive user information that is logged. However, ProTracer provides no resilience or self-protection measures.

3.5.4 Summary of Context Correlation

This section presented several context correlation approaches that have been discussed according to the requirements of an IDS provided at the beginning of this chapter. Not all requirements were explicitly investigated in the original papers. Thus, some discussion details are open to interpretation. Anyhow, Table 3.4 summarizes the overall results of this discussion. The

detection approaches were classified according to their assistance scope when it comes to intrusion detection specifically in the network. Approaches either

- complement the context of particular network events ([Sna+91], [AL01]) with information from the hosts,
- correlate intrusion data from different sources ([Aba+03], [Pei+16]) that describes the same attack, or
- perform endpoint security ([MZX16]) that can be used for network forensics.

The approaches in this section primarily perform no intrusion detection themselves but are meant to assist in the intrusion detection process. This is reflected in the assessments of the overall detection accuracy. The host context from [MZX16] can be of great value for the intrusion detection but the approach itself does not even sketch a detector. In contrast, the algorithm in [Aba+03] is a direct blueprint for the correlation of different data sources to come up with a detection result. The other approaches at least highlight their help for network intrusion detection. However, apart from such showcases that might be used to demonstrate a concrete detection scenario, the approaches' general utilization for detection is more important for this section. Thus, the remaining of this summary assesses the approaches' concepts regarding assisting in satisfying the requirements for intrusion detection.

Almost all others cannot assist in detecting stealthy attacks with long time periods between malicious events. Most approaches have not mechanism in place to timeout event information but still linking new to historical data. The flow-oriented approaches generally fail to link context between different network flows, but [AL01] can at least assist in stealthy attacks that encrypt their malicious traffic or obfuscate it my similar means. Only [MZX16] can effectively reveal historical relations because it reflects the life cycle of system objects to out-date event information and additionally propagates the provenance information transitively among the objects until it is logged.

None of the approaches can effectively assist in detecting distributed attacks. But all of them are universally applicable as they provide context that can be for intrusion detection in various ways and for multiple purposes. Satisfying the requirement of an attack representation is, however, highly dispersed across the approaches. While the majority just reports the full details, [Sna+91] reports the path an originating user is path hopping through the network and [MZX16] reports only the those provenance information that make an significant change to the system.

The event-based processing and reporting enables almost all approaches to be real-time capable. Only [Pei+16] performs on batches of log data which leads to the analysis results being only available at the end of each batch. The result compared to [MZX16], however, gives a more detailed and accurate insight into relations host events for a particular attack.

The approach [Pei+16] is assessed worst regarding the efficiency requirement. This is because the creation of the log graph requires to compared the features among all log entries which results in a quadratic complexity. All other approaches are more efficient and most of them fulfill this requirement completely. The fulfillment of the scalability requirement, however, is not given by any but one approach. This one approach [MZX16] is also only partly covering scalability because it leverages parallel processing only on the hosts themselves but not for the main task that is performed centrally. Also the privacy is not fully addresses by any of the approaches.

Context Correlation for Network Intrusion Detection					
Requirements	*DIDS* [Sna+91]	*Transaction Inspection* [AL01]	*Anomaly Correlation* [Aba+03]	*HERCULE* [Pei+16]	*ProTracer* [MZX16]
Detection Accuracy (R1)	Ø	Ø	✓	Ø	✗
- Stealthy Attacks	✗	Ø	✗	✗	✓
- Distributed Attacks	Ø	✗	✗	Ø	✗
- Attack Representation	✓	✗	✗	✗	✓
- Universal Applicability	✓	✓	✓	✓	✓
Real-Time Detection (R2)	✓	✓	✓	✗	✓
Efficiency (R3)	Ø	✓	✓	✗	✓
Scalability (R4)	✗	✗	✗	✗	Ø
Easy Deployment (R5)	Ø	✗	Ø	Ø	✗
Resilience and Self-protection (R6)	Ø	Ø	Ø	Ø	Ø
Privacy (R7)	Ø	✗	Ø	✗	Ø

Table 3.4: Summary of context correlation approaches regarding their compliance with the requirements given at the beginning of this chapter. Checkmark symbols ✓ indicate the fulfillment of these requirements, crossing symbols ✗ their non-fulfillment, average symbol Ø their partial match.

That most data for the approaches in this section is retrieved from the many hosts in the network results in two consequences. First, this provides some basic resilience against internal failures because the main task can still be performed with degraded accuracy even when data from few hosts is missing. However, none of the approaches applies resilience measures in the data processing itself. Second, a deployment is not easy because it requires the active participation of hosts. And the deployment of [AL01] and [MZX16] is even harder because a modification of the operating system (OS) or application is required to get access to the data instead of leveraging already existing data interfaces.

3.6 Multi-Step Attack Reconstruction

This section presents approaches for the detection of mutli-step attacks, i.e., those that are not targeting at multiple hosts at the same time but attacks that achieve their goal in a sequence of different activities. The detection approaches here leverage all raw and already correlated data from the intrusion detection process to reconstruct the individual actions performed by the attacker.

The approaches are structured into two groups (cf. Section 3.2.2). The first group contains approaches that reconstruct that path that an attacker took during lateral movement [Nia+20]. Approaches in the second group focuses on the detection and interconnection of the attack steps to reveal the full attack scenario.

3.6.1 Attack Path Reconstruction

The following paragraphs discuss approaches that reconstruct the sequence exploited systems during an attacks.

Alert Correlation with Attack Graphs In their work on Alert Correlation with Attack Graphs (ACAG), Roschke et al. [RCM11] extend the original work from Wang et al. [WLJ06] on attack graphss (AGs) for alert correlation. The basic idea is to model attack steps an attacker can perform in the network to achieve a specific goal [Sch99], e.g., to gain access to a specific system or to make the system unavailable to others. In their work, Roschke et al. have a special focus on the hosts and the sequence in which they are exploited. The authors formalize the detection that basically requires to map alerts to the AG.

The alert correlation with attack graphs requires only a few alert attributes, including the timestamp, source host, destination host, and classification of the alert. The AG, however, requires to be modeled with expert knowledge about vulnerabilities and their effects. The intention of the AG is to hold specific information about the different steps that an attacker must perform for the attack to be successful. Intuitively, there is not a single sequence of steps, but different combinations of steps might be possible, also depending on the attack surface. All the different attack possibilities are encoded in the AG. For that, a vertex in the AG reflects a specific attack step as a triple $v = (im, h, r)$ with the impact im, the affected host h, and the vulnerability reference r of the attack step (cf. MulVAL [OGA05]).

The correlation algorithm itself maps alerts to specific nodes in the AG. For that, alerts and nodes must satisfy some restrictions regarding their features. The authors identified three important alert features and how they are mapped to a vertex v in the AG: (1) the alert classification maps to the vulnerability reference r of v, (2) the alert source host maps to the host of another node v' in the AG, and (3) the destination host maps to the host of node v. In total, Roschke et al. give several combinations of these three features to restrict the mapping of alerts to nodes. When mapping alerts, the algorithm is identifying and merging duplicate alerts by equal feature values and close timestamps.

The merged alerts that have been matched to a node in the AG, are transferred into a dependency graph (DG) and reflected as vertices. An edge between two alerts in the DG exists, if there is a relation between the respective mapped vertices in the AG. The authors identified three kinds of relations in the AG that will lead to an edge in the DG. Intuitively, such an edge requires the alerts to be in sequence towards forming a path in the AG, i.e., towards a sequence of steps that result in a successful attack. The last step in correlation is to search for the most interesting alert sequences. For that, the authors leverage the Floyd Warshall algorithm [Flo62; War62] to find all the shortest paths. With that, they identify the longest shortest paths, i.e., the sequences of dependable alerts that reflect all steps in the DG and therefore indicate a successful attack.

In summary, Roschke et al. identify sequences of alerts that match an attack defined in the AG. This is primarily based on the IP addresses of the source and destination hosts in the alerts and

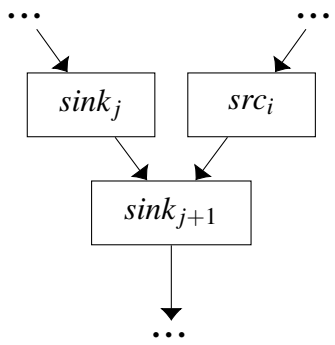
the IP address of the host in AG vertices, among others. The result of the ACAG is basically the sequence of hosts that are involved in the attack. Because of further information like the vulnerability reference and the attack impact, the extracted alert path in the DG contains some more hints about details in every attack step.

These extracted paths are a good representation and abstraction of the multi-step attack. It is universally applicable for the detection of any multi-step attack that is specified in the AG. Parallel actions cannot be specified in the AG for the detection of distributed attacks, instead the approach is most effective when modeling combinations of different actions. Depending on how long alert information remains mapped to the AG, stealthy attacks might be detected. The actual detection with the transformation into the DG and the extraction of paths, however, is triggered manually and, therefore, not real-time capable. The extracted paths might only report the few targeted hosts or services and abstract more sensitive information.

The alert correlation with attack graphs is efficient because a new alert must be mapped against a fixed set of nodes in the AG, which results in the complexity $\mathcal{O}(|V|)$ where $|V|$ is the number of nodes. However, the correlation does not scale. Furthermore, the approach is not resilient against failures or overload situations. Integration of the correlation itself is easy with access to the alert of an already deployed IDS. But the initial creation of the AG and its maintenance must be customized for each environment and requires some effort.

ZePro Sun et al. [Sun+16] propose ZePro, a probabilistic approach for the detecting of multi-step attacks that might contain zero-day exploits. When attackers perform multiple steps, the authors note that most of the exploits used are non-zero-day that can be detected. But in case the attacker uses a zero-day exploit in its attack path, this exploit can still be linked by a chain with other non-zero-day exploits to reveal the full path. ZePro detects such zero-day attack paths in two steps. First, ZePro creates an *object instance graph* from host data to capture the information flow among system objects, i.e., processes, files, and sockets, also across hosts. Second, a Bayesian network (BN) [Pea85; Pea88] incorporates intrusion evidence collected from various information sources, i.e., intrusion alerts, to capture intrusion propagation among the objects and to compute the most likely attack path the attacker took through the network.

The actual detection is conducted on a central machine that collects the system calls on monitored hosts. During an offline analysis, this data is first transformed into a so-called *object instance graph*. This is based on the key insight, that an intrusion propagates from one system object, i.e., process, file, or socket, to another object through system calls such as *read* and *write*. These system calls highlight the information flows among the objects and therefore they are denoted as source and sink. In contrast to provenance graphs (cf. Section 2.3.2), each node in ZePro is not a system object $o_x \in O$, but the i^{th} instance of the object in a specific time-dependent version. This versioning allows to encode the sequence of interactions with the object o_x, i.e., via system calls, and therefore decouples them when they are in fact not dependent, i.e., reading a file only depends on writes that happened previously. The result is a DAG $G = (V, E)$ with nodes V being the system object instances $src_i, sink_j, i, j \geq 1$ where src_i and $sink_j$ are the i^{th} instance of the system call source and the j^{th} instance of the system call sink, respectively. A new instance, i.e., version, of an object is created only if it is the sink in the system call. The edges E label the kind of *call dependency* according to the system call between the most recent versions of the source and the new version of the sink. In addition, an edge is added between the previous and the new version of the sink to label the *state transition dependency* (cf. Figure 3.7).



			$sink_{j+1}$	
			infected	uninfected
$sink_j$ = uninfected	src_i	infected	τ	$1-\tau$
		uninfected	ρ	$1-\rho$
$sink_j$ = infected	src_i	infected	1	0
		uninfected	1	0

Figure 3.7: The infection propagation model and CPT at node $sink_{j+1}$.

After parsing the system calls, an *instance-graph-based BN* is constructed from the nodes and edges in the object instance graph. In general, a BN consists of nodes that represent a variable of interest and directed edges that denote the causality relations between two nodes. The strength of such a causality relation is indicated using a conditional probability table (CPT). In ZePro, the intention of the BN is now to label each node with either `infected` or `uninfected`. This is done using an infection propagation model to state the probability for each node of being infected (cf. Figure 3.7). If not already infected, the label of a node depends on the source src_i in the respective system call that leads to the new sink version $sink_{j+1}$. If the source is already considered infected, the intrusion is propagated to the sink with a *contact infection rate* τ. However, the sink can also become infected for other reasons, which is reflected by the *intrinsic infection rate* ρ. Although $0 \leq \tau, \rho \leq 1$, Sun et al. propose to set ρ to a very small constant number, whereas τ should be tuned by security experts. Once infected, objects stay infected in any future versions until recovery operations are performed to clean the infected system.

Apart from this infection propagation model, ZePro can incorporate intrusion evidence into the instance-graph-based BN in two ways, i.e., towards initially labeling a specific object instance as infected. If the infected state is detected with enough certainty or confirmed manually, the objected instance is immediately labeled to be infected. Otherwise, the local observation model (LOM) by Xie et al. [Xie+10] is leveraged to model observations of potential malicious activities. In this case, the observation is added as a direct child node of the object instance in question. The CPT of the observation then inherently indicates the false rates with respect to the state of the real object instance.

After probability inference, each node in the instance graph receives a probability of being infected. To reveal multi-step attack paths, ZePro extracts all relevant nodes and the edges among them. Intuitively, this includes all nodes with high infection probabilities, e.g., above 80%. But also nodes that have both an ancestor and descendant with high infection probabilities to allow for zero-day exploits in the model. The actual extraction is based on a depth-first search (DFS) in the instance graph.

The result of ZePro is a path among the system objects to detail how the intrusion propagated. The attack path can even be reconstructed across multiple hosts when establishing a network-wide object instance graph that connects the socket objects of across monitored hosts. The detection is based on a BN that quantitatively computes the probabilities of object instances being infected. Connected through dependency relations, the instances with high infection probabilities form a path, which can be viewed as a zero-day attack path.

The quality of the instance graph is good and accurately reflects the ground truth of system objects because their versioning and their causal dependencies among each other. However, the system is not efficient because of the large overhead for creating the instance graph. The actual detection based on the BN does not scale and is also not real-time capable as this is triggered manually. Furthermore, the detection accuracy depends on the quality of the IDS alerts that initially set the intrusion evidence that propagates through the BN. ZePro in general is applicability to many attack scenarios but particularly fails to detect distributed attacks that cannot be modeled by a path through the graph. But ZePro is eventually able to detect stealthy attacks, depending on the time span for which intrusion evidence remains in the instance graph. Anyway, the outcome is representing the multi-step attack by its path with characteristic information about the system objects involved.

The privacy of users is threatened by collecting all system calls, but only the affected system calls are included in the reported path. Requiring to collect this host data in the first place, prevents an easy deployment. But no more expert knowledge is further needed in operation, apart from the IDS alerts. Because of the centralized processing, ZePro is not resilient, nor can it adapt to overload situations.

3.6.2 Attack Scenario Reconstruction

The following paragraphs discuss approaches that reconstruct attack scenarios that consist out of a variety of different attack actions.

Fuzzy Attack Patterns Daneshgar et al. [FA16] present the concept of fuzzy attack patterns that are historically relevant alerts that frequently occur in consecutive attack steps. The authors use this concept to cluster alerts to so-called fuzzy attack events that are not only based on feature similarity but also based on combinations of alerts that frequently occur together. The authors propose an online model for this alert correlation to identify such related alerts in real-time continuously from an input stream of alerts. The algorithm incorporates mechanisms to identify complex scenarios as usually found in multi-step attacks. The system is composed of two modules. The first module emits fuzzy attack events that are reported to the security operator. The second module also consumes these events to mine fuzzy attack patterns, i.e., frequent patterns of alerts, that are fed back to the first module to improve clustering.

The first module is named *online incremental fuzzy clustering module*. Its goal is to cluster a stream of alerts to fuzzy events. In particular, a fuzzy event created at timestamp t and is defined as $e = \{t, \{a_1, \ldots, a_n\}, \{m_1, \ldots, m_n\}, w\}$ with the clustered alerts $\{a_1, \ldots, a_n\}$ and a measure named membership degree $m_i \in [0;1]$ for each alert a_i that describes how good the alert is related to previous alerts in the fuzzy event. Both lists might be extended online by adding new alerts to the event. In addition, every fuzzy event e_j is associated with a time-dependable weight w_j that is increased by m_i at the time an alert a_i is added and that is otherwise decreased by the fading function $f(t) = \alpha^{-at}$ until another event is added. The fuzzy event is said to be *stable* when its weight falls below a threshold θ or when its lifetime exceeds a threshold $maxtl$. Once a fuzzy event becomes stable, it is removed from the clustering module and added to the output queue of fuzzy attack events.

The actual clustering in the first module, i.e., deciding which events e_j a new alert a_i should be added to, is based on three criteria that are individually calculated and averaged to derive

the membership degree m_i for an alert a_i to a fuzzy event e_j. The first criterion is the similarity between IP addresses. For both source and destination IP address in the alert a_i, a match is searched among the union of source and destination IP addresses of all alerts in e_j. The second criterion is the time similarity between the time of the new alert a_i and the time of the last alert in the existing event e_j. The similarity is 1 if the time difference $diff$ is within a duration threshold and otherwise continuously decreases by the function $1-\alpha^{-diff}$. The third criterion is the similarity between the alert types. For that, the highest similarity between the alert type of a_i and the types of all alerts in e_j is chosen. The idea for this is based on the original work by Zhu et al. [ZG06]. Zhu et al. use neural network approaches, Multilayer Perceptron (MLP) and support vector machine (SVM) in particular, to learn the relation among alert types as they occur in multi-step scenarios. In contrast to them, Daneshgar et al. eliminate the need for training phases or expert knowledge but learn the so-called *Correlation Strength* between alerts online based on historical data in an unsupervised fashion. Based on these three similarity criteria, the alert a_i is added to any e_j for that the membership degree is above the threshold ε_{Mem}. If not added to at least one fuzzy event, a new one is created containing the alert. The next module becomes active when a fuzzy events becomes stable and added to the output event queue.

The second module is named *fuzzy frequent inter event pattern mining module*. Its goal is to extract frequent fuzzy patterns from the queue of fuzzy attack events and to update the correlation strength matrix (CSM) (cf. Zhu et al. [ZG06]). State about previous fuzzy events is maintained in a so-called inter event pattern (IEP)-tree. Intuitively, frequent patterns indicate the alerts that statistically happened together. When having the events $e_1 = \{t_1, \{A, D, E\}, \dots\}$, $e_2 = \{t_2, \{B, E, C\}, \dots\}$, and $e_3 = \{t_3, \{D, E\}, \dots\}$, one can derive that alert E frequently happens after alert D. This mining happens in three steps that are performed upon every fuzzy attack event emitted by the first module:

1. Tree-Construction: The tree is initialized with a root that stays empty all the time. As the tree is built by adding child and sibling nodes, each path from the root towards a leaf node indicates a length-l pattern with l being the number of alerts in the pattern. Based on the work by Chiu et al. [Chi+11], the alerts reported in fuzzy attack events first result in length-1 patterns associated with their number of occurrences. Once a length-1 pattern becomes frequent, i.e., occurs more often than a threshold, the pattern can be extended to a length-2 pattern by appending another alert that frequently follows. The following alert is with respect to one of two dimensions, either consecutive within the same event of within the alerts of a consecutive event. The new pattern is then added as a child of the respective length-1 pattern.

2. Tree-Traversing: When creating new length-l patterns, they trigger updates of the Correlation Strength Matrix. However, only alerts from different fuzzy attack events are selected to propagate the correlation of the respective alert types. This makes the Correlation Strength Matrix to reflect relations between consecutive steps in a multi-step attack. Based on the membership degree m_i of the alerts in the respective patterns, the correlation strength value is updated.

3. Tree-Pruning: The last step out-date alerts because they happened a while ago. This must also be reflected in the IEP-tree to be able to adapt to new emerging alert patterns. However, instead of deleting the affected nodes in the tree completely, only their counter is decreased when the respective alert times out. The authors implement this by a sliding window over the last N emitted fuzzy attack events. New events that get inserted into the queue result in increasing the counter of the contained alerts while removed alerts result in

decreasing. Only if the counter falls below the threshold for frequent patterns, any pattern containing this alert is removed.

The outcome of the alert correlation algorithm are fuzzy attack events, i.e., alert clusters, that continuously adapt to new emerging attack scenarios. Instead of clustering alerts to aggregations that describe a single attack step by similar alert features, the clusters based on fuzzy attack patterns describe alerts related to the same attack that might be composed out of several steps.

Because of the event-based and continuous identification of fuzzy attack events, the detection theoretically works in real-time. In practice the real-time requirement is only partly covered because the reporting of attack events is delayed by the fading function for their weights. These events represent the whole attack, but do not structure the attack according to the different steps. Because of the three different criteria used to correlate alerts, the detection covers most attacks and is universally applicable. Especially the criterion of IP similarity allows to identify distributed attacks. Also stealthy attacks are partly able to detect because the fuzzy attack event of a slow but ongoing attack will evolve until the event becomes stable.

The detection itself, i.e., identifying fuzzy attack events, is efficient because checking a new alert against each existing attack event according to the three criteria is done by simple operations like intersection of sets or numerical difference. However, updating the CSM through the identification of frequent patterns is an additional overhead. But as this can be skipped, e.g., in overload situations, while still ensuring reasonable detection accuracy, the system is party resilient. No alerts are filtered and will finally go into an attack event, which is not privacy friendly but leads to all alert details to be reported. Despite the complex design of the correlation, the system is easy to deploy because it processes existing IDS alerts and even adapts to new attack scenarios automatically because of the frequent pattern mining.

HOLMES Milajerdi et al. [Mil+19] propose HOLMES, a system for the detection of advanced persistent threats (APTs). The authors note that APTs follow an APT lifecycle model that is also known as the cyber-kill-chain [HCA11]. This model describes the different stages an APT attacker usually performs, e.g., the initial compromise, establishing a foothold, escalating privileges, to complete its mission. HOLMES leverages this model for the correlation between suspicious information flows to produce a detection signal that indicates the presence of a coordinated set of activities that are part of an APT campaign. The detection is conducted in three steps:

1. Generating alerts from low-level events that are significant to an APT behavior
2. Correlating alerts with multiple activities of the attacker to indicate an ongoing APT attack
3. Presenting the attack scenario by an intuitive summary to an analyst

The events in the first step, i.e., the alert generation, are based on raw audit data from `auditd` for Linux, `dtrace` for BSD, and `ETW` for Windows. From this audit data, a provenance graph (cf. Section 2.3.2) $G_P = (V, E)$ is created, where nodes V represent processes as subjects and files, pipes, and sockets as objects. Edges E reflect the kind of dependencies among subjects and objects, i.e., event names like *read* or *write*, among others. Nodes in the graph are versioned and a new version of a subject or object is created when a new incoming edge is added, i.e., when the dependencies of a node change. To detect significant events that might represent APT behavior, the authors leverage the tactics, techniques, and procedures (TTPs) as defined in the MITRE's ATT&CK framework [Str+18] when mapping low-level events to APT stages. In

HOLMES, a TTP specification abstracts a particular behavior such as making memory regions executable. A TTP, therefore, summarizes the respective audit events of the different OSes that can be used to achieve this behavior and assigns them the APT stage in which the behavior is usually seen. Mapping system calls to high-level specifications of malicious behavior this way, results in alerts that are meaningful to the remaining of the APT detection.

In the second step, i.e., the alert correlation, the relation among these potential APT alerts must be analyzed for ongoing APT attacks. For this, HOLMES leverages a concept known as *prerequisites* and *consequences* [NCR02]. In general, this links two entities when the consequences of the one satisfy the prerequisites of the other. In HOLMES, the TTPs define prerequisites as rules, that reflect requirements to other TTPs that are used in the previous APT stage. Intuitively, that two APT stages succeed according to the cyber-kill-chain is reflected by the prerequisites. And therefore, two related APT stages are detected if there is a match of the prerequisites for two alerts.

In the last step, i.e., the attack scenario representation, HOLMES extracts alerts that correspond to a series of consecutive APT stages, primarily driven by the prerequisites as defined in the TTPs. For the representation to human beings, a High-level Scenario Graph (HSG) $G_{HSG} = (A, D)$ is constructed. Nodes A in the HSG represent alerts that can be related to an ongoing APT attack, while the edges D represent information flow and causality dependencies among the alerts. To reduce the false positive HSGs, the authors make HOLMES to learn benign patterns and create heuristics that assign weights to nodes and paths in the G_{HSG} based on the severity of alerts. The latter allows to present the HSGs to the analyst ranked according to their priority. The authors of HOLMES suggest assigning TTP severity based on the Common Attack Pattern Enumeration and Classification (CAPEC) [MIT].

In summary, HOLMES's detection is based on an APT's most essential high-level behavioral steps and the information flow dependencies between these steps. The dependencies are reconstructed from host audit data. The result is an attack scenario that describes attack steps following the cyber-kill-chain model. The presentation in the HSG is informational and a good abstraction of the whole attack, including a prioritization based on severity. The detection has a universal applicability to all attack scenarios because they all leave traces in the utilized host data. As the focus is on a series of actions following the APT model, distributed attacks are not reflected. Stealthy attacks are not addressed in the sense of large time spans between consecutive actions but the approach reveals such attacks that locally mimic benign behavior.

The detection algorithm seems to perform mostly efficient because of simple matches and transformations, but the overhead if high because of the volume of system calls to process. Similarly, although the system calls are received in an event-based fashion, it is not convincing that all three steps are executed upon every event. More likely, the creation of the HSG is triggered manually, which causes HOLMES to only partly work in real-time.

HOLMES does not provide any measures for scalability or resilience. Despite the usage of sensitive user information included in the system calls, the detection result as HSG abstracts is limited to relevant actions only and furthermore abstract their details, which is privacy friendly. Deploying HOLMES is not easy because it requires the active participation of hosts to allow collecting the system calls. Furthermore, an extensive knowledge base is needed to model APT attack behavior.

SAM Meier et al. [MSK05] propose the Signature Analysis Module (SAM) for IDSes to enable them to efficiently check for multi-step signatures [Mei04]. Referring to the Event Description Language (EDL) [Sch04] that is based on colored Petri nets [Jen87], Meier et al. present their modeling approach with multi-step signatures and five strategies to efficiently leverage them for signature matching.

Signatures in the EDL are modeled with *places* $P_i \in \mathscr{P}$ and *transitions* $T_j \in \mathscr{T}$, comparable to nodes and directed edges in a graph. In this Petri net like approach, a signature starts with an *initial place* and ends either with an *exit place* or *escape place* that represent the full signature match or an aborted match, respectively. Transitions represent events that trigger state change from one place to another. They connect these initial and termination places to form paths with *interior places* in between. Signature checking is performed using tokens that represent signature instances and unconditionally spawn at the *initial place*. Following outgoing transitions might lead to spawning tokens at successor places and eventually reaching the *exit place*. For conditional traversal through the net, places $\mathscr{P}$ define features and transitions $\mathscr{P}$ define conditions that have to match to fire the transition. If a place defines the feature `userID` the tokens at this place are assigned the value of a particular ID. An outgoing transaction could require the event type to be `FileCreate` and the event user to match the place user.

When using these Petri net like signatures in a naive way, it would require to check an observed event $e \in E$ against all signatures $\mathscr{S}$, the contained transitions $\mathscr{T}$, and their conditions $\mathscr{C}$. This does not scale well with the increasing number of incomplete signature instances, i.e., tokens. Because of that, SAM incorporates five matching strategies that build upon each other:

1. Strategy 1 (Transition Types): As transitions require events of a particular type, transitions can be indexed in a key-value fashion such that the type of an event is efficiently mapped to a list of transitions.

2. Strategy 2 (Intra-Event Conditions): The so-called intra-event conditions are comparisons between event features and constants, i.e., independent from place features or token values, respectively. For every transition, these conditions have to be evaluated only once per event. If successful, the inter-event conditions with input places and their tokens are evaluated.

3. Strategy 3 (Token Values): Instead of evaluating the inter-conditions of an transition for an event with each token at the input places separately, tokens can be indexed by their value for efficient value matching. If a place defines multiple features, also multiple indexes exist, each mapping to lists of tokens. Then, each feature is looked up efficiently and the intersection of mapped tokens determines the matching tokens.

4. Strategy 4 (Common Sub-Expressions): While Strategy 2 and 3 already account for redundant evaluations of the same condition for a single transaction, common sub-expressions might sill exist in the condition blocks of different transitions. SAM identifies such common sub-expressions and calculates their boolean value only once for each event. The result of true or false is used across the evaluation of all transitions without calculating the value again. In this regard, the authors highlight standard techniques for identifying common sub-expressions [ASU88].

5. Strategy 5 (Condition Prioritization): As final optimization, the different evaluations of conditions can be ordered such that a mismatch is detected at an early stage and that the whole transition cannot fire. This avoids to evaluate remaining and more expensive

condition. The authors note that conditions can be prioritized statically or dynamically but should be optimized for failure to most effective.

In summary, SAM models multi-step signatures based on high-level Petri-nets using the signature specification language EDL. SAM incorporates strategies to linearly scale with the number of events to check against the complex signatures. This allows to efficiently detect multi-step attacks in a signature-based approach. The EDL can model various attack scenarios that are then implemented by SAM for universal applicability. This also includes stealthy attacks because a token representing an ongoing attack stays in the Petri-net until its detection is confirmed or aborted. But distributed attacks are not further focused in the detection. Once a signature matches by reaching the exit place, SAM provides no better abstraction than the details on all relevant features and conditions, which is too detailed for a reasonable attack representation.

SAM works in real-time by checking for any transitions and signature matches in the Petri-nets upon a new event. This event processing with the central Petri-nets does not scale and is not resilient against obfuscated data or component failures. The events, furthermore, include full sensitive details also when presenting the detection results, which violated the user's privacy. Because the events also come from hosts and therefore their active participation is required, the deployment of SAM is not as simple as with passive network monitoring.

3.6.3 Summary of Multi-Step Attack Reconstruction

This section presented several approaches for the reconstruction of multi-step attacks that have been discussed according to the requirements of an IDS provided at the beginning of this chapter. Not all requirements were explicitly investigated in the original papers. Thus, some discussion details are open to interpretation. Anyhow, Table 3.5 summarizes the overall results of this discussion. The detection approaches were classified according to the focus of the reconstruction which is on either

- the attack path through the affected systems ([RCM11], [Sun+16]) or
- the attack scenario ([FA16], [Mil+19], [MSK05]) by interconnecting the different attack steps.

The overall detection accuracy is assessed according to attacks in the intended scope of the approaches. While all approaches can detect multi-step attacks, two particularly suffer from different limitations. The scenario steps in the attack graphs of [RCM11] usually describe more higher level steps and can hardly be modeled accurately enough to be mapped with technical IDS alerts. Despite the mix of different correlation criteria in [FA16], alert similarity and frequent alert patterns are likely to not be enough to find inconspicuous and rare alert relations from multi-step attacks. Better accuracy is achieved by the remaining three approaches. The causal relations among the system objects from system calls in [Sun+16] avoid any false positives in comparison to relations based on feature similarity. And the BN models the propagation of infections reasonably based on the accurate object relations. The detection accuracy is also high in [Mil+19] and [MSK05] because their descriptions of multi-step attack scenarios are generic enough to be sound while being technical enough to be incorporated by an algorithm without any ambiguousness.

Only the approach based on Petri-nets with their tokens in [MSK05] support the detection of stealthy attacks by keeping attack scenarios open until their detection is confirmed or aborted.

Multi-Step Attack Reconstruction					
	Path		Scenario		
Requirements	*ACAG* [RCM11]	*ZePro* [Sun+16]	*Fuzzy Attack Patterns* [FA16]	*HOLMES* [Mil+19]	*SAM* [MSK05]
Detection Accuracy (R1)	Ø	✓	Ø	✓	✓
- Stealthy Attacks	Ø	Ø	Ø	Ø	✓
- Distributed Attacks	✗	✗	Ø	✗	✗
- Attack Representation	✓	✓	Ø	✓	✗
- Universal Applicability	✓	Ø	✓	✓	✓
Real-Time Detection (R2)	✗	✗	Ø	Ø	✓
Efficiency (R3)	✓	✗	Ø	Ø	✓
Scalability (R4)	✗	✗	✗	✗	✗
Easy Deployment (R5)	Ø	Ø	✓	Ø	Ø
Resilience and Self-protection (R6)	✗	✗	Ø	✗	✗
Privacy (R7)	Ø	Ø	✗	✓	✗

Table 3.5: Summary of multi-step attack reconstruction approaches regarding their compliance with the requirements given at the beginning of this chapter. Checkmark symbols ✓ indicate the fulfillment of these requirements, crossing symbols ✗ their non-fulfillment, average symbol Ø their partial match.

Unfortunately, the same approach is assessed worst regarding the attack representation because the detected scenario is reported by giving full details that are related to the attack. This is better handled for example by [Mil+19] where the attacks are visualized in a graph, filtered using benign patterns, and ranked according to the severity of alerts.

Regarding distributed attacks, only [FA16] can partly detect them. This is because it uses basic similarity of alert features, among others, which groups alerts from or to different hosts when some features stay equal. Regarding the universal applicability, only [Sun+16] is too narrowed towards identifying the path of attack actions. The other approaches link attack activity in a more generic way, such that they potentially can also be used for other purposes than linking consecutive attacker actions to multi-step attacks.

The two approaches [FA16] and [MSK05] both implement an event-based correlation to report detected attacks. However, [FA16] is not fully real-time capable because this approach always waits for more alerts that might belong to the attack. In contrast, [MSK05] immediately reports the attacks as soon as the last event regarding a scenario signature is processed. This event-based processing in combination with well suited data structures for comparisons makes this approach also one of the most efficient ones. There is only [RCM11] with a similar efficiency based on

similar comparisons. However, none of the approaches are scalable as they all perform the detection centrally.

The deployment of most approach is not easy. They either exclusively process data from hosts and require their active participation ([Sun+16], [Mil+19]) or they require extensive expert knowledge to implement the detection depending on the environment or scenario ([RCM11], [MSK05]). Only the retrieval of existing IDS alerts and their self-adapting correlation as in [FA16] can be easily deployed. This is also the only approach that partly covers the resilience requirement as the detection relies on different criteria. This on the one hand allows to continue operation if one criterion fails and on the other hand allows to skip the most computational intensive criterion when under load.

With respect to privacy, [Mil+19] is assessed best because the reported attack is characterized by its attack scenario and described by the tactics used in each step. This abstracts sensitive data in this higher-lever report and only includes the most necessary information related to the attack.

3.7 Summary

This chapter has introduced the requirements of an IDS and analyzed the state of the art for detecting attacks that eventually affect large portions of the monitored network. Most of the approaches have been identified as unsuitable to implement the full intrusion detection process.

In a first step, the well-established practice of using SIEM systems for managing security information from different machines, services, and functions in the network has been discussed. SIEM system provide an effective way to perform simple data processing and visualization of monitoring metrics and security analysis. Nevertheless, they usually perform the analysis centrally and do not scale. Also, the analysis capabilities are usually limited to data aggregation, their timestamp-based correlation, and pattern detection. This gives high-level performance and security indicators but lacks of fine-grained information that is required to detect and report other than simple security incidents. This is furthermore highlighted by the lack of any network traffic information that actually goes beyond the traffic volume.

Approaches to detect network-wide attacks aim to identify parallel malicious actions of the same kind as in distributed attacks, e.g., scans, DDoS, or botnets, where multiple hosts are involved in the same attack step. For that, the approaches not only classify the monitored actions or events as malicious but also group those that are related to the same attack. Scenario-specific approaches are in general most accurate by implementing a detector that eventually reports the attack as soon as the scenario definition is confirmed in the data. In contrast, more general approaches are based on some definition of similarity among the events or alerts to identify and correlate predominant attributes. This is more universally applicable but these approaches can only report the identified patterns which are then up to the security operator to interpret. Altogether, these approaches can only correlate related events when their common ground is directly visible in the monitored network traffic. Otherwise, putting together the pieces of the attack requires an extended monitoring scope.

Approaches for context correlation aim for a larger scope of monitoring to get a better vision on and understanding of particular actions in the network. These approaches provide additional context to assist in intrusion detection. The general problem with these approaches is that they usually do not perform intrusion detection on their own. Instead, they sketch a concept or

framework to correlate data from different domains. In best case, such an approach correlates two independent detection results for better overall accuracy. But in most cases, the approaches only correlate events without any classification for maliciousness. The actual detection algorithm is out of scope then.

Moreover, this chapter has taken a closer look at approaches for the detection of multi-step attacks. In contrast to distributed attacks, different actions build upon each other and are performed by the attacker sequentially, most likely involving different systems in each step. An interesting part of these approaches is that they can causally reconstruct the chain of steps by interconnecting them with strong evidence based on either flows in the network or system calls on the hosts. This focus on the linking of steps, however, has also some drawbacks. The missing insight into the actual steps causes not all related actions and consequences within this step to become visible, which in return has severe consequences when it comes to attack mitigation and recovering.

Concluding, there is no one-fits-all solution to establish an accurate, efficient, and easy to deploy IDS. Too many different kinds of attacks exist that cannot be covered by a single detector. Thus, different approaches have to be selected along the intrusion detection process, while each approach must overcome the challenges to accurately fulfill its specific purpose. Apart from that, the selection of approaches must ensure a compatibility so that they effectively leverage each other. Some approaches seem promising by following a strategy that is similar to the intrusion detection process. PAC uses meta-alerts to first identify directly related actions and to then interconnect them further to multi-step attacks. Fuzzy Attack Patterns incorporates different correlation criteria to account for alerts both directly related and part of a multi-step attack.

4 Security Monitoring

Intrusion detection suffers from restricted visibility, which causes false classification of events or events to be missed by the monitoring altogether. To strengthen the accuracy of intrusion detection, the visibility in security monitoring as a whole must be extended. This chapter highlights two different strategies to mitigate the problem of restricted visibility by combining monitored events for classification. The correlation of log files, as done by security information and event management (SIEM) systems (cf. Section 3.3), however, is not sufficient to increase the accuracy of intrusion detection because the SIEM data is too coarse-grained. Instead, the two event correlation strategies in this chapter work even independently from each other so that they can be combined to enhance security monitoring.

The first mitigation strategy extends the monitoring scope with respect to the supervised system. Implementing this strategy, Section 4.1 correlates host and network data in real-time for a joint monitoring. The result is high-quality monitoring data that highlights the link between host and network events, which forensic investigations but also approaches for intrusion detection can leverage. While this first strategy extends the monitoring visibility with heterogeneous data across different domains, the second strategy correlates homogeneous events from the same domain and directly leverages it for intrusion detection. Particularly for the network domain, this chapter provides two examples that correlate the communication relations among hosts in a larger network to detect certain attack scenario targeting the network at large. The detection examples on the basis of the communication graph resulting from correlated network flows include the attack scenarios of distributed scan campaigns in Section 4.2 and peer-to-peer (P2P) botnets in Section 4.3.

4.1 Joint Host and Network Monitoring with zeek-osquery

An intrusion detection system (IDS) can analyze network traffic for signs of attacks and intrusions. However, encrypted communication limits the IDS's visibility, and sophisticated attackers additionally try to evade their detection. To overcome these limitations, this section presents an approach to extend the scope of a network intrusion detection system (NIDS) with additional data from the hosts. Implementing this concept results in the integrated open-source *zeek-osquery* platform that combines the Zeek IDS with the osquery host monitor. This platform collects, processes, and correlates host and network data at large scale, e.g., to attribute network flows to processes and users. The platform can be flexibly extended with custom detection scripts using host data that is both already correlated and additionally dynamically retrieved. A distributed deployment enables it to scale with an arbitrary number of osquery hosts.

This section revises parts of the conference paper [HSF20]. The section first introduces a formal model to correlate host and network events for attributing network communication to users and applications. Afterward, a scalable system, i.e., zeek-osquery, is presented to collect host events and to correlate them with network events in real-time. Last, this section discusses detecting attack scenarios through the correlation of host and network events.

4.1.1 Combining Host and Network Monitoring

Connecting host and network data, which is currently done with SIEM systems, extends the monitoring visibility. However, compared to SIEM systems, a more fine-grained correlation is required for the detection of sophisticated attackers and an interactive data retrieval is more efficient and privacy friendly. To achieve the required extended visibility based on regular network monitoring, the idea is to determine related context that come from the hosts, e.g., user or process information. Network-related host activity is identified as the missing piece between both domains. Incorporated into the following model, it introduces the attribution of traffic in the network to applications and users on the respective hosts.

4.1.1.1 Model for Host and Network Events

An event e describes an action on the host or in the network. While a low-level event on the host could be the start of a process, accessing a file, or authenticating a user, the only low-level event in the network is the transmission of individual packets. In general, low-level events include any information that is directly observed with the action, e.g., the name of the started process, the access mode to a file, or the username used in authentication as well as the headers and payload in a network packet.

Apart from such low-level events, also meta events exist that result from processing other low-level or meta-events. Information in raw-level events is meant to be neutral and isolated from other events. In contrast, particularly network events can be interpreted on different layers. A Transmission Control Protocol (TCP)-SYN packet on session layer indicates as a connection attempt and a Hypertext Transfer Protocol (HTTP) request on application layer indicates the access to a particular resource on the web server. Also, the combination of events can result in meta events, e.g., when the status code in an HTTP response matches a previous HTTP request, or when assembling packets that belong to the same TCP connection to track its life cycle according to the TCP state machine.

In general, an event e describes as a set of attributes $A_{<type>}$ that depend on the $type$ of the event. Its definition applies to both low-level events and meta-events:

$$e_{<type>} := \{a_i \in A_{<type>}\}$$

The following paragraphs detail the network and host events and their types that are relevant to attribute network flows.

Network Events Network events are triggered when hosts communicate in the network. In today's computer networks, communication is based on the IP protocol. Thus, a host has at least one IP address and might use a port P to communicate via a protocol like $prot \in \{TCP, UDP, \dots\}$. A network event is always tied to a specific packet but is possibly also tied to a higher layer in the Internet model, e.g., network, transport, or application layer.

The model for network events distinguishes between two major event types. Both are based on the network 5-tuple that consists of two IP addresses, two ports, and a protocol to identify the communication between two hosts.

Events related to a network *packet* are directly direction aware, so the IP addresses reflect the *source* and *destination* of the respective packet. Thus, packet events have at least the following attributes:

$$e_{packet} := (IP_{src},\ P_{src},\ IP_{dst},\ P_{dst},\ prot,\ \ldots)$$

In contrast, other events are related to a network *flow* that reflects consecutive packets on a higher Internet layer. The flow terminology denotes the flow initiator as the *originator* and the other host as the *responder*. Thus, flow events have at least the following attributes:

$$e_{flow} := (IP_{orig},\ P_{orig},\ IP_{resp},\ P_{resp},\ prot,\ \ldots)$$

Note that the IP addresses and ports in two related e_{packet} and e_{flow} events are actually the same. However, the *source* IP and port corresponds to the *originator* only when flow and packet are in the same direction. Otherwise, the packet *source* corresponds to the flow *responder* as in a message reply.

Network-related Host Events Host events arise from actions on hosts regarding a large number of different properties, including processes, files, users, software, and devices, among others. Apart from the host identifier h, objects are keyed with more attributes to identify the object and to describe the action.

By design, today's major operating systems (OSs) identify *processes* by a process ID (*pid*). Furthermore, processes use *sockets* to send and receive network packets. The identifier of a socket is a unique ID, i.e., the combination of a file descriptor (fd) and *pid*.

A process event $e_{process}$ contains the process ID *pid*, the *path* of the binary, the process ID of the parent process $ppid$, the user ID *uid* and potentially more attributes:

$$e_{process} := (h,\ pid,\ path,\ ppid,\ uid,\ \ldots)$$

A socket event e_{socket} contains the process ID *pid* and the file descriptor fd. Analogous to a network flow, a socket abstracts a series of packets. Thus, a socket event additionally includes the attributes of the respective network 5-tuple. However, sockets abstract the communication from the view of a particular host. Therefore, the tuple denotes the IP addresses and ports with *local* and *remote* according to their direction. Thus, a socket event is defined as:

$$e_{socket} := (h,\ pid,\ fd,\ IP_{loc},\ P_{loc},\ IP_{rem},\ P_{rem},\ prot,\ldots)$$

In practice, OSes usually provides two kernel interfaces to retrieve process and socket events. Particularly for Linux, this is:

- *Kernel audit*: Several system calls (syscalls) exist for the interaction of processes with the kernel, including `execve` to spawn processes as well as `bind` and `connect` to establish incoming and outgoing flows, respectively. The kernel audit in Linux enables the direct retrieval of process and socket events by the monitoring of such syscalls.
- *Kernal status*: The kernel status data holds all attributes about current processes and sockets. Probing this status enables retrieving processes and sockets but not directly leads to events that indicate changes.

Interface	Resource	IP_{loc}	P_{loc}	IP_{rem}	P_{rem}	*prot*	Retrieval
Kernel audit	bind syscall	?	?	no	no	TCP	push
	connect syscall	no	no	yes	yes	TCP	push
Kernel status	/proc/net procfs	yes	yes	yes	yes	all	pull

Table 4.1: Properties of socket events, depending on the interface and resource in Linux.

The data from both, *kernel audit* and *kernel status*, has different properties, as summarized in Table 4.1 using the example for socket events in Linux. While the audit asynchronously pushes new socket events, the status has to be frequently probed and compared with the previous one to detect socket events. If a short-living socket starts and ends in between the probing interval, the pull-based data retrieval from the kernel status misses respective socket events. However, also the push-based kernel audit has a major disadvantage. Events stemming from audit include only the parameters that syscall monitoring was able to capture, i.e., parameters that have been explicitly set by the caller of the syscall. Thus, some relevant attributes might be missed in respective events. For example, in case of a `connect`, only the destination IP address and port are part of the call. The local IP address and port remain unknown, even when combining several socket-related syscalls. Similar restrictions apply to the bind syscall, for which Table 4.1 has the question mark indicating that the local IP and port is available only when explicitly set by the syscall or 0.0.0.0 and 0 otherwise when dynamically chosen.

Other Host Event Relations An essential aspect of sockets and how processes use them to send and receive packets is that processes can depend on other processes. Assume a worm that spreads by exploiting a vulnerable program via crafted network packets. After the process is taken over, the worm spawns more processes to scan for potential victims and to spread by sending crafted packets. The process that receives the initial exploit packet might spawn a child process that, in the aftermath, sends exploit packets to more victims. For detection and forensic reasons, it is crucial to identify the relation among these two processes. The same need to identify the relation among processes to reconstruct relations among network flows is given by the example of Secure Shell (SSH) chaining [ZP00]. An attacker abuses a SSH server as jump host to obfuscate the original source, known as stepping stone attack [WRW02; WR03; WR10]. Although network-based timing analysis can detect the relation of the respective network flows with uncertainty, incorporating the process relations from the host perspective can lead to a better detection accuracy [CKT05].

The necessity for linking processes in a joint monitoring is more generally illustrated in Figure 4.1. It uses the notion $isParent(pid_1, pid_2) \in \{0,1\}$ for two processes pid_1 and pid_2 on a host $h \in H$ to indicate that pid_1 is the direct or indirect parent process of pid_2 or that they are the same. A network message from host h_1 goes to h_3 via the proxy host h_2. A network monitor alone would see two unrelated network flows between h_1 and h_2 as well as h_2 and h_3. Only when incorporating the process relations on host h_2, the big picture becomes visible. Because the receiving process pid_{21} is the parent of the sending process pid_{22}, the full path of the message via h_2 is revealed. This knowledge is essential, especially in the case of worms, SSH hopping, or lateral movement of attackers. Other indirect relations between two processes, e.g., via files, can be tracked using more detailed process interactions such as in [Bat+15]. For demonstrating the capabilities of joint monitoring, the remaining of this section uses the notation of direct process relations only.

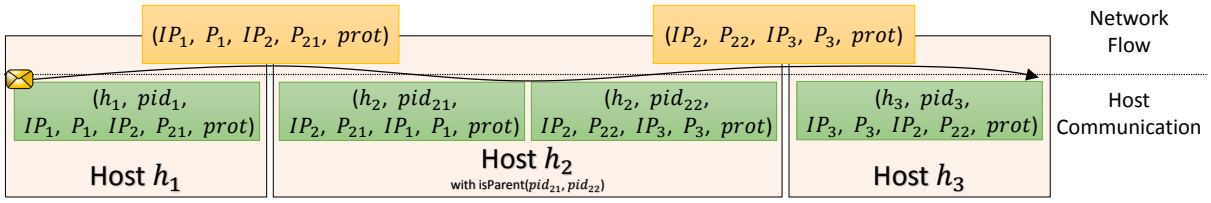


Figure 4.1: Host communication and process relations across different hosts.

4.1.1.2 Attributing Network Flows

The attribution of a network flow to the originating process is always conducted with respect to the originator or the responder host (cf. Section 4.1.1.1). It requires the identification of the respective socket on both hosts, where applicable. For simplicity, identifying the socket for attribution is sufficient, as this is the missing link to other host activity, e.g., processes and files [Bat+15]. The attribution relies on the distinct network 5-tuple, i.e., the respective five attributes source IP, source port, destination IP, destination port, and protocol. These attributes are part of both events representing a network flow e_{flow} or socket e_{socket} (cf. Section 4.1.1.1).

The identification of the respective socket for a network flow requires a match on the network 5-tuple between the socket and network flow. On the originator, this is a socket representing an outgoing flow, and an incoming flow on the responder. Distinguishing the direction is relevant as the 5-tuple for sockets reflects the IP address and port of the local and remote hosts, while it reflects originator and responder host for network flow. On the originator host, remote IP and port in the socket 5-tuple of the outgoing flow have to match responder IP and port of the respective network 5-tuple. On the responder, remote IP and port in the socket 5-tuple of the incoming flow have to match originator IP and port of the respective network 5-tuple. Figure 4.2 illustrates the originator with its outgoing socket (top left), the responder with its incoming socket (bottom right), and the network flow with its full 5-tuple (middle).

In contrast to kernel status events, the events from kernel audit render a match on the source of a network flow impossible, because the respective socket attributes are missed (cf. 4.1). To account for that, sockets and network flows are correlated for attribution according to Figure 4.2 and the following three steps:

(1) Identify originator and responder hosts by the IP addresses in the network flow. This requires a maintained list of IP addresses and hosts in the network.

(2) On the originator and responder, identify the socket(s) for which the flow destination equals the remote or local socket info, respectively.

(3) Also, require the flow source to equal local or remote socket info, respectively.

The correlation is unambiguous when the socket attributes for step 3 are available. Otherwise, the correlation might be *vague*. In case of two hosts with a `connect` syscall to the same destination IP address and port, the correlation is still unambiguous because of step 1. However, it is vague for the same host with multiple flows to the same remote IP and port (from different source ports). Ideally, the correlation outcome is exactly one socket for the originator and the responder. However, in case of a vague correlation, the correlation might output more then one candidate sockets. It is then unknown, which of these candidate sockets is the correct one for the attribution.

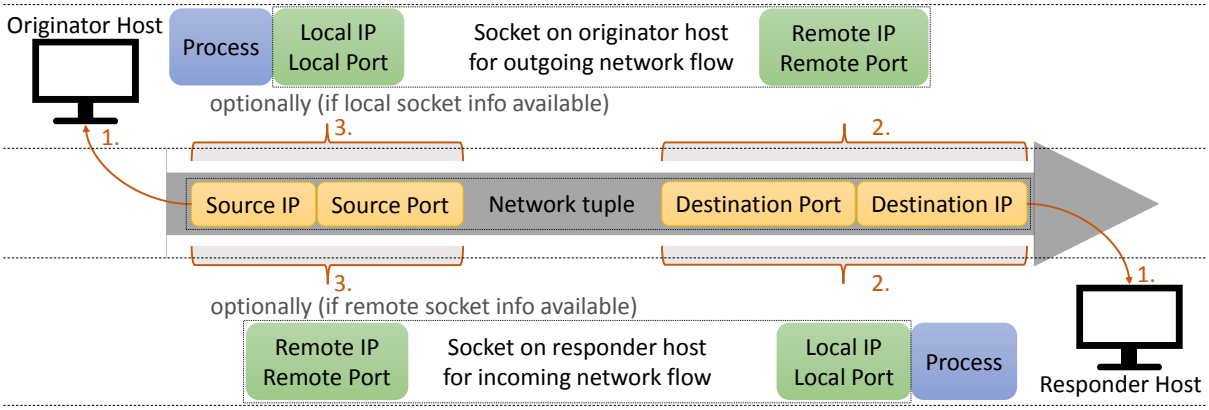


Figure 4.2: Attributing a network flow to a process on originator and responder host by matching the network tuple to the socket information on each of the hosts.

4.1.1.3 Validity of Host Events

Constant host activity including new processes and sockets grows the amount of retrieved kernel audit and status events by time. Apart from potential storage and space constraints, maintaining the data regarding its *validity* is critical to the attribution accuracy. Processes and sockets that already terminated some time ago must not be considered for the attribution of a currently ongoing network flow. For that, a *state* stores those host events that are currently valid, e.g., beginning with the creation of a process and ending with its termination. Thus, a state maintains the host events according to their validity, aiming to follow the life cycle of processes and sockets from their creation to termination.

Data from kernel status can be easily incorporated into the state, as the status reflects a current snapshot, and comparing it to the previous snapshot allows identifying new or removed processes and sockets, respectively. For the kernel audit, the life cycle for processes and sockets can be followed by syscalls like `execve`, `socket`, or `close`. This works as long as the process decides on its own to terminate or to close a socket. However, there will not be such an audit event when the process crashes or the TCP connection breaks. To prevent the state from being polluted in such cases, it regularly requires to perform a *verification*. This is done by probing the host if all the processes (pid) and sockets (pid, fd) in the state are still present by the current kernel status.

4.1.2 Monitoring with zeek-osquery

The rest of this section describes `zeek-osquery`, a system for the collection, analysis, and correlation of host and network data that follows the approach from Section 4.1.1 for a joint host and network monitoring with extended visibility. Here, the system itself based on the existing monitoring tools Zeek (cf. Section 2.4.1) and osquery (cf. Section 2.4.2) is introduced, explaining the scalable system design regarding communication and data processing.

4.1.2.1 Overview and Concept

The system zeek-osquery for joint host and network monitoring deploys two different monitoring tools. The first tool is Zeek that monitors and analyzes the communication in the network, usually

at a central location in the network, e.g., at a core router or at the upstream to the Internet. The second tool, the host monitor osquery, runs on multiple hosts in the network to provide detailed information about the host and its OS state via a SQL-based interface. To achieve a joint monitoring, Zeek retrieves host events e_{host} from osquery hosts. The communication between both tools is realized by the Zeek-internal event-dispatcher and communication library Broker[1]. While Zeek has Broker built-in, osquery is extended by Broker to enable the communication. To retrieve host events, Zeek sends SQL queries to osquery to subscribe to particular events such as new processes. When osquery captures a new process, a respective event is sent to Zeek, where any host event is analogously processed to network events natively. Hence, Zeek is now a platform with the ability to correlate various information from both host and network events.

All the logic for retrieving host events and processing them is implemented in a novel Zeek framework. This collection of Zeek scripts allows to easily manage osquery hosts in the network, query the hosts for particular events, and correlate the data in event handlers. As one correlation example, this thesis elaborates on attributing network connections (cf. Section 4.1.1.2) that works on the basis of bridging the host-network divide [FMN05] and further enables new capabilities for intrusion detection.

The whole system has been designed with efficiency and scalability in mind. In addition to osquery continuously sending new host events to Zeek for logging and further analysis, Zeek can query osquery tables on-demand to retrieve a snapshot of an osquery table, e.g., to gather additional data about the host upon a suspicious action. This enables an interactive analysis of hosts that reduces the amount of collected host data but still allows to investigate a specific security issue. For large deployments, the Zeek correlation platform can be set up in a distributed manner with multiple, communicating Zeek instances.

4.1.2.2 System Architecture

On the Zeek side, a novel Zeek *framework*, i.e., a collection of Zeek scripts, enables the basic interaction with osquery hosts. The communication is realized by connecting all osquery hosts and Zeek nodes via a publish-subscribe overlay established via Broker. Distinct topic names, labeled as *groups* throughout this thesis, address specific osquery hosts or groups of them in the overlay. Apart from some default groups, custom ones can either be pre-configured on hosts or dynamically controlled by Zeek. It uses group labels to control the SQL queries for specific selections of osquery hosts. An *interest* denotes the binding of a query to a group. It contains additional information, e.g., whether the query is executed regularly or just once and how to send the results back to Zeek. This way, Zeek can publish an interest over Broker to osquery hosts in a particular group, e.g., for logged in users on all monitored servers. The new Zeek framework enables Zeek to be capable of:

- requesting complete results to a one-time query immediately,
- scheduling queries that are regularly executed,
- removing queries from the execution schedule,
- making osquery hosts to join a group,
- making osquery hosts to leave a group.

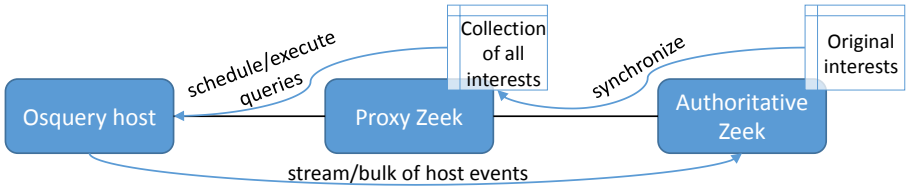


Figure 4.3: Communication among osquery and Zeek.

When Zeek publishes an interest via the overlay, all currently connected osquery hosts will receive this interest and start to continuously reply with new events until Zeek revokes this interest. While this works for osquery hosts that are connected at the time of originally publishing the interest, others that join the overlay later cannot know about previously published but valid interests. However, osquery hosts just joining the overlay must be aware of currently active interests, i.e., previously published but valid interests. Thus, Zeek has to publish current interests again but to the joining host only this time. As managing the joining hosts and resending current interests to them introduces some additional overhead, zeek-osquery outsources this task to a special proxy Zeek. All the others tasks regarding requesting and processing host events is implement by an authoritative Zeek. These two roles can be carried out by a single Zeek instance or can be separated among two Zeek instances, as illustrated in Figure 4.3:

- An *authoritative* Zeek is an instance that is the origin of an interest or a group. This role defines queries and is responsible for processing any query results, i.e., host events, from the osquery hosts.
- A *proxy* Zeek is an instance that accepts connections from osquery hosts and holds state about them. Furthermore, it keeps a collection of all currently active interests and groups that originate from any authoritative Zeek. This way, the proxy Zeek keeps state for hosts that join later and will tell them all currently active interest without putting any additional load on the authoritative Zeek.

Apart from meeting the concerns regarding osquery hosts that join the overlay later, the communication design including different Zeek roles enables a distributed deployment that scales when adding more Zeek instances (cf. Section 4.1.2.3). However, before presenting the distributed and scalable deployment, the following paragraphs detail the novel Zeek framework with the two Zeek roles regarding requesting and processing host events.

Requesting Host Events When an authoritative Zeek defines an interest or group, it is flooded in the overlay to all proxy Zeek instances. They store all current interests and groups from any authoritative Zeek until the originating Zeek revokes or disconnects. Optionally, proxy Zeek instances can act as a cache for interests and groups if an authoritative Zeek disconnects for a short period, e.g., because of technical failure. Without this caching, proxies would immediately update their collection and make osquery to remove respective queries from the schedule. This is considered unnecessary if the authoritative Zeek comes up with the same interests a second later, again resulting in updating the collection of proxies and modifying the schedule of osquery. For this reason, the proxy Zeek can delay the removal of interests and groups from its collection when the authoritative Zeek disconnects. If the original interests and groups are re-published within a grace timeout, they will become stay active in the collection without any changes to scheduled queries on the hosts.

1. https://docs.zeek.org/projects/broker

```
event zeek_init() {
  # Make hosts in the subnet to join the group
  osquery::join(134.100.0.0/16, "UHH");
  # Define the interest and its query
  local interest = [$ev=host_processes,
                    $query="SELECT pid,name,path,
                      cmdline,uid,parent FROM processes"];
  # Make host group to regularly execute the query
  osquery::subscribe(interest, "UHH");
}
```

Listing 4.1: Snippet using the novel zeek-osquery framework in Zeek scripting language to define a group and an interest.

```
event host_processes(resultInfo:osquery::ResultInfo,
        pid:int, name:string, path:string,
        cmdline:string, uid:int, parent:int) {
  # Define Log mapping
  local info: Info = [$t=network_time(),
                      $host=resultInfo$host,
                      $utype=resultInfo$utype,
                      $pid = pid,
                      $name = name,
                      $path = path,
                      $cmdline = cmdline,
                      $uid = uid,
                      $parent = parent];

  # Write log entry to file
  Log::write(LOG, info);
}
```

Listing 4.2: Snippet using the novel zeek-osquery framework in Zeek scripting language to handle (log) query results.

When an authoritative Zeek uses the interest and group application programming interface (API) of the osquery framework in Zeek, the framework makes sure that interest and group definitions are maintained until they are revoked by calling another API. This includes that connected osquery hosts are informed about any changes in group and interest definitions as soon as the change happens. It also includes that new osquery hosts are instructed about current groups and queries as soon as the host is connected. An example script is given in Listing 4.1 that queries the processes of all hosts that have an IP within the subnet 134.100.0.0/16.

Processing Host Events Because of the flexible publish-subscribe overlay via Broker, osquery hosts can establish a Broker connection to any proxy Zeek. A successful connection will automatically trigger an application-level handshake initiated by the osquery host to announce itself to Zeek. This handshake is required so that the proxy Zeek can hold state about its directly connected osquery hosts. After a successful announcement, the proxy Zeek is in control of the directly connected osquery host. This means that the osquery host from now on accepts and executes any received interest. It is the responsibility of the proxy Zeek to filter and forward only applicable interests to its directly connected osquery hosts. When the connection between the

osquery host and the proxy Zeek terminates or breaks, both nodes clear their respective internal connection state.

Although the proxy Zeek controls directly connected osquery hosts, note that query results are forwarded back to the originating authoritative Zeek by default, which can be controlled by the response topic used in the Broker overlay. Upon receiving the responses, the definition of an appropriate event handler in the osquery framework is required. Listing 4.2 is such an event handler as Zeek script snippet to log information about new and terminated processes using the log file mechanisms of Zeek. Every handler for host events includes a `ResultInfo` object as first parameter that gives general information about the event, e.g., from which osquery host, a reference to the original query, and whether the event reflects a row added to or removed from the osquery table. The other parameters of the handler correspond to the requested columns of the osquery table.

While Listings 4.1 and 4.2 just document some APIs of the Zeek framework for the interaction with osquery hosts, there is more interaction happening in the background. This includes for examples that the proxy Zeek automatically retrieves the IP addresses of all its directly connected osquery hosts. The IP addresses are required so that the proxy Zeek can organize the hosts into groups based on IP addresses. Thus, to keep its own state about directly connected osquery hosts up-to-date, a proxy Zeek itself publishes some interests to these hosts in addition to the collection of interests from authoritative Zeeks.

4.1.2.3 Distributed and Scalable Deployment

The overlay communication can be separated in two groups: (1) Communication between Zeek instances, denoted as Zeek *backend*, and (2) communication between osquery hosts and the Zeek backend. In this design, the proxy Zeeks belong to the backend and expose themselves as an intermediary between authoritative Zeeks and the osquery hosts (cf. Figure 4.3). The proxy Zeek holds a collection of all interests originally published by authoritative Zeeks. A synchronization is required to keep the collection up-to-date. Based on its collection, the proxy Zeek schedules queries that should be executed on a specific osquery host. The query results are returned directly to the authoritative Zeeks as host events.

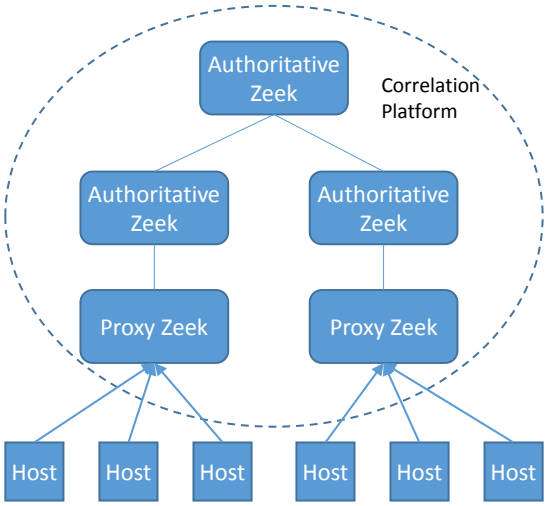
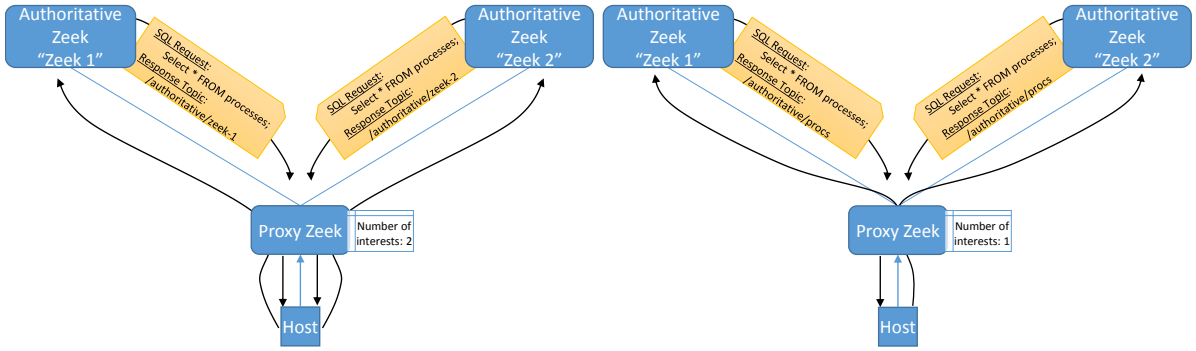


Figure 4.4: Distributed setup with multiple Zeek instances.



(a) Individual response topic for every Zeek. (b) Equal response topics for equal SQL queries.

Figure 4.5: A query example with multiple Zeeks, in which the choice of response topics can decrease the network load regarding number of transmitted host events.

The Zeek instances in the backend interconnect in a hierarchical structure, as illustrated in Figure 4.4. Distributing the load among multiple Zeek instances can be achieved in three ways:

- Resource intensive correlation tasks can run exclusively on an additional authoritative Zeek instance. For that, another Zeek joins the overlay and publishes interests for events that are required for detection. The resources of this instance are solely available to the detection, and all other instances can continue using their resources to perform their tasks.
- If large amounts of osquery hosts would overwhelm a single authoritative Zeek instance, the osquery hosts are organized in groups, with one out of multiple Zeek instances being responsible for one group. All Zeek instances then publish interests for the same query but to specific groups.
- To reduce the load on a single proxy Zeek instance, multiple instances can be deployed. Then, each one needs to handle a lower number of directly connected hosts that still receive the same interests as before.

After an osquery host joined the overlay, it is managed by the directly connected proxy Zeek and accepts and executes any forwarded interest. However, note that interests originally come from authoritative Zeeks and query results should also be routed back to them via the publish-subscribe overlay. For that, the originating authoritative Zeek by default sets the response topic to its own topic when publishing interests. If the same interest query originates from multiple Zeeks, it is suggested to choose the same response topic across Zeek instances for the same interest. This way, equal interests can be consolidated on the proxy Zeek, and the query results are sent over Broker only once to multiple authoritative Zeek instances at the same time. Figure 4.5 compares these two cases, in which two authoritative Zeeks send a interest for the same SQL query. On the left side in Figure 4.5a, they choose individual response topics. Consequently, the proxy handles the two interests as two different ones. In contrast, the Zeeks choose the same response topic in Figure 4.5b. Consequently, the effectively equal interests are handled as a single one on the proxy. Thus, the osquery host executes the respective query only ones and sends every response event only once. Broker then takes care of efficiently routing this message to both Zeeks, e.g., similar to IP multicast [Dee89].

Next, the architecture of the novel Zeek framework for requesting, receiving, and processing host data is presented. More precisely, this framework includes scripts that enable the correlation

of host data. It is referred to with the term processing pipeline. Any authoritative Zeek runs this pipeline to perform a correlation of the received host data.

4.1.3 Event Correlation for Network Attribution

The outcome at the end of the processing pipeline, as part of the novel Zeek framework, is the correlation of host and network events (cf. Section 4.1.1). It is Zeek-typically implemented event-based to allow custom scripts to reuse events that are emitted within the pipeline. On this basis, it is easily possible to create additional scripts to process those events further.

The three different stages in the processing pipeline are illustrated in Figure 4.6 and are detailed in the remainder of this section.

Querying The first stage defines interests, i.e., SQL queries, that are sent to and scheduled on osquery hosts. Events on this stage are a continuous stream of raw host events that directly come from osquery. Thus, incoming events reflect updates of an osquery table, e.g., processes, users, or sockets (cf. Section 4.1.1.1). Custom scripts can reuse these raw host events for different purposes, including logging them to disk, performing state-less analysis of individual events, or forwarding them to the next stage for processing in a stateful manner.

State This stage assembles raw host events from osquery to states in real-time. A state reflects the current host status reconstructed from all previous events (cf. Section 4.1.1.3). Thus, every new event potentially updates the state by both adding and removing state information, i.e., a process is added upon its creation and removed upon its termination. Updating requires host events to report changes regarding the status of a host in the sense of new, modified, and removed information. As regular tables in osquery allow to query for new, modified, and removed information, Zeek can accurately reconstruct state based on these tables. However, the so-called event-based tables often do not include removed information. Because of that, the state is verified periodically. For that, the correlation platform utilizes one-time queries to retrieve the state information that is not valid anymore from osquery.

In this stage, the framework emits events whenever the state changes. This stage is suited to merge raw host events from different tables in case they describe the same class of data. For example, information about sockets on a host can be found across the tables *socket_events*, *listening_ports*, and *process_open_sockets*. However, events from all three tables are merged to reflect a common state about host sockets.

Furthermore, holding state in the correlation platform about host information allows browsing them efficiently without interacting with the hosts every time. This reduces the load on osquery hosts and provides state information to other scripts.

Correlation Correlating two events is done based on a common attribute of them. More explicitly, not only events, but also state information can be correlated. However, the correlation is always triggered by an event. Such a trigger can be a raw host event that directly comes from osquery, a state event that indicates a state change, or a network event that comes from Zeek itself.

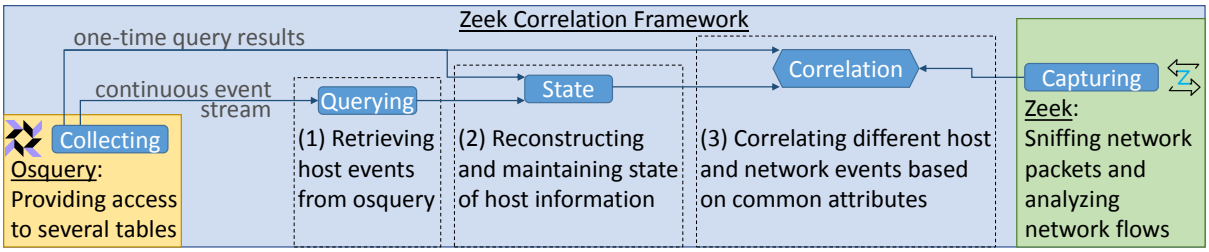


Figure 4.6: Architecture of the processing pipeline.

As an example, an attribution is implemented to link network flows with the respective application and user in real-time (cf. Section 4.1.1.2). For demonstrating the effect, the statistics about every network flow in Zeek (`conn.log`) is extended to list the respective host, application, and user additionally.

4.1.4 Scenario Detection with zeek-osquery

To demonstrate the new capabilities of intrusion detection using zeek-osquery (cf. Section 4.1.2), the detection of three attacks scenarios is implemented into the processing pipeline (cf. Section 4.1.3). All three examples make use of correlating host and network data in real-time.

Execution of Internet Files Sharing indicators of compromise is an effective way to secure against known malware but also to investigate past attacks. Such an indicator could be the origin of a file on the Internet, i.e., an Uniform Resource Locator (URL). The following detection highlights the example scenario of a user executing a file downloaded from an URL in an email.

1. *Advertisement*: The user receives an email with a link to download a file from a website. This advertisement to the file, however, is an optional step and might not apply to all executions of Internet files. The email could also have a file directly attached and is downloaded together with the mail from the mail server.

2. *Download*: A user starts a download on a host either without any preceding action or more likely by following an advertisement, as described in step 1. Anyhow, the result is the user downloading a specific file from a specific origin, e.g., a web or mail server.

3. *Execution*: The user executes the downloaded file, either from where he saved it to, or the file is directly executed from its temporary location.

If Zeek can inspect emails and web traffic, it notices the reference to the origin, e.g., the link or attachment in an incoming mail via Simple Mail Transfer Protocol (SMTP). Zeek, furthermore, keeps track of accessing this email by noticing the host from which the mail was retrieved via Post Office Protocol version 3 (POP3) or Internet Message Access Protocol (IMAP) (step 1). By monitoring the network data stream the very same host, Zeek detects the download of the file from the origin in question to this host, on which the mail containing the origin was retrieved (step 2). Instead of keeping a copy of the file, Zeek needs only to remember its hash. Furthermore, all executed binaries on the host are reported to Zeek (step 3) and are then compared with the hash of the downloaded file to detect its execution.

```
event process_binary_hash(
        resultInfo: osquery::ResultInfo,
        md5: string) {
  # Known smtp attachment hashes only
  if (md5 !in smtp_hashes) { return; }

  # Get mail recipient(s)
  print fmt("Execution of email attachment for '%s' on host '%s'",
    smtp_hashes[md5], resultInfo$host);
}

event process_state_added(host_id:string,
        process_info: osquery::ProcessInfo) {
  # One-time query to retrieve process binary hash
  local query_string = fmt("SELECT md5 FROM hash WHERE path=\"%s\"",
    process_info$path);
  local query = [$ev=process_binary_hash,
          $query=query_string];
  osquery::execute(query, host_id);
}
```

Listing 4.3: Snippet using the novel zeek-osquery framework in Zeek scripting language to detect the execution of SMTP attachments.

The detection of executed Internet files if demonstrated by the Zeek script in Listing 4.3 for the execution of mail attachments. For that, the script analyzes incoming mails via SMTP and maintains a list of hashes of executable file attachments together with the mail recipients. By using the proposed processing pipeline for host-network correlation, the script leverages the existing continuous stream of host events to get notified about new processes. Upon new entries in the process state, the detection script queries the respective host for the hash of the executed binary using the on-demand query capabilities of zeek-osquery. If the returned hash matches that of a downloaded mail attachments, the detection identified the execution of a mail attachment.

SSH Hopping A good practice in network design is to have isolated subnets, e.g., for different departments, offices, and goods production. For security reasons, direct communication between these subnets is usually prohibited and restricted by a firewall. However, exceptions exist for administration or business processes. In such a case, only specific hosts are allowed to communicate through the firewall. A security problem arises when one of these exceptional hosts is taken over and communicates in behave of a third host. Then, the compromised host is abused as a jump host to enable unauthorized access to the isolated subnet by redirecting traffic from the third host.

One possible way to realize this attack is the so-called SSH hopping or SSH chaining. It presumes an attacker with valid SSH credentials, e.g., obtained through social engineering or performed by an insider attacker. The attacker logs in into the first machine via SSH and uses it as a jump host to establish further SSH connections into another subnet. In many cases, this poses a violation of the security policy and should be detected. Although regular network monitoring can detect incoming and outgoing SSH connections on the same host, it cannot verify whether both are related.

zeek-osquery can detect SSH hopping as follows: First, The detection script keeps state of ongoing SSH sessions by leveraging native Zeek events to detect successful SSH logins. By leveraging the flow attribution, the process on the originator and responder of the SSH session is identified where applicable. Second, the script also browses all other currently ongoing SSH sessions on the originator and responder hosts. It detects an SSH hopping on one of the hosts when there is

- a pair of incoming and outgoing SSH connections to and from the same host,
- where the process for the outgoing connection is a subprocess of the process for the incoming connection.

The latter condition is checked using the relation *isParent* from Section 4.1.1. It is implemented by an on-demand query that asks the host for all parent processes of the process for the outgoing SSH connection. The script matches the parent processes with processes for incoming SSH connections. In the case of an intersection, this not only verifies the SSH hopping but also reveals the user that logged in.

TLS Interception In recent years, the need for encrypted communication became stronger to protect privacy but also to prevent eavesdropping on business secrets. However, almost all versions of Transport Layer Security (TLS), as used for secure web browsing via Hypertext Transfer Protocol Secure (HTTPS), were by their design vulnerable to attacks (Poodle, BEAST, CRIME, BREACH, DROWN, and recently ROBOT) or did not prevent downgrade attacks such as FREAK and Logjam. The latest TLS standard, version 1.3, is meant to be secure against all previous TLS attacks and puts additional security measurements in place like authenticated encryption with associated data (AEAD), forward secrecy, and ephemeral keys. While privacy benefits from the new standard, intrusion detection with deep packet inspection is not possible anymore as TLS interception is practically impossible.

As only participants in possession of the session keys can decrypt the traffic, an IDS like Zeek would also be required to retrieve these keys to decrypt the traffic and to look into it. Recently, a new Linux kernel feature, named in-kernel TLS (kTLS), was introduced. It allows hosts to perform the de- and encryption of established TLS sessions in the kernel for performance reasons. For that, a user space application provides the cryptographic material, i.e., session keys and parameters, to the kernel using the `setsockopt` system call. To leverage this, an osquery table monitors the respective system call and extracts the material. Utilizing the new processing pipeline, Zeek retrieves and maintains the keys of ongoing TLS sessions. Currently, there is no live decryption implemented, but the mapping of sessions and their keys which is saved for forensic reasons.

While kTLS is not widely used yet, cryptography libraries like openssl are already developing support for it [Fou20]. Using kTLS for traffic decryption in intrusion detection might be a promising solution in certain environments. However, since kTLS is not yet adopted by common applications, this theoretical scenario is not pursued in this thesis.

4.1.5 Summary of Joint Monitoring

To compensate the restricted visibility of a NIDS, the monitoring and intrusion detection platform *zeek-osquery* with extended visibility has been introduced. This platform allows for a joint

monitoring of hosts and the network and a fine-grained correlation for the monitoring data. This approach solves the problem of the NIDS going blind because of traffic encryption and other attacking techniques that try to evade IDS detection. With the extended visibility by hosts, the NIDS mitigates the negative effects on intrusion detection accuracy that stem from going blind. The mitigation generally works this way that the additional host data complements network data that is missing relevant information because it is either out of network scope or becoming unavailable because of traffic encryption.

The zeek-osquery platform itself is centered around the Zeek monitoring and IDS tool. *Zeek* natively sniffs and analyzes network traffic, but originally lacks host context. A novel Zeek framework extends the analysis in Zeek to retrieve and incorporate host data that is provided through the host sensor *osquery*. A common channel between Zeek and osquery has been developed to enable Zeek to request host data and osquery to serve it. In particular, the complementing host data provides additional context for individual flows in the network. zeek-osquery attributes network flows to the respective users and applications running on the host. The necessary correlation of heterogeneous monitoring data from hosts and the network must be performed in real-time for accurate results. The accuracy and the operation of zeek-osquery in real-time is investigated in Section 6.3.1.

In a large-scale deployment of zeek-osquery, a single centrally deployed Zeek would not be capable of processing the host data that comes from osquery running on all the hosts in the network. Because of that, the whole platform must be efficient and scalable. These concerns are supposed to be met by an overlay that connects the osquery hosts and a distributed variant of the correlation platform with multiple Zeek nodes. In this overlay, Zeek nodes can share responsibilities regarding handling osquery hosts and processing their data. While this already reduces the load on individual Zeek nodes significantly, an efficient messaging schema additionally minimizes the overhead. The efficiency of zeek-osquery is evaluated in Section 6.3.2.

The goal of the new monitoring capabilities with zeek-osquery is to generally extend the network visibility because of the additional host data. The network attribution, however, is only a preliminary example that is performed during monitoring for the fine-grained correlation of host and network data. Custom detection algorithms in Zeek can benefit from the resulting visibility by leveraging the correlated monitoring data. A demonstration of the new detection capabilities includes the scenarios of executing Internet files and SSH hopping. In addition, the new concept of hosts assisting in the inspecting of encrypted network traffic is presented. Section 6.3.3 investigates to which extent the new monitoring capabilities with zeek-osquery allow for detecting such intrusion scenarios.

4.2 Scan Campaign Detection

Every targeted attack starts with reconnaissance to identify victims. A widespread technique for such reconnaissance is port scanning to learn about accessible services that can eventually be exploited. Thus, reconnaissance helps the attackers to identify potential victims that expose a specific attack surface in favor of the attackers. In fact, massive port scan activity is expected to hit any network that is connected to the Internet. Although not causing any harm directly, detecting these scans allows the victims to anticipate upcoming attack steps. However, the sheer flood of scan activity overwhelms security operations centers (SOCs) and renders a

manual analysis of the scan activity impossible. Furthermore, distributed scans as part of a scan campaign are even harder to analyze or detect in the first place.

This section presents a detection algorithm to identify related scan activity and summarize it as scan campaigns. This is particularly challenging because multiple scanners scan collaboratively in such a scan campaign. That these scanners are controlled by the same attacker, however, not directly apparent. Thus, SOC would detect every single source individually, with each seeming to scan only a few hosts and ports. Thus, the algorithm in this section identifies scanners that collaborate in scan campaign by finding similarities in the scan activity among different scanners.

This section revises parts of the conference paper [HWF20]. The foundations for this work have been acquired through a supervised bachelor thesis [Reg19]. The section first models the attacker in the scenario of a distributed scan campaign. Then, it describes the first part of the algorithm for detecting scan campaigns with its characterization of the scan activity. This is followed by describing second part that describes the clustering of scanners with similar behavior into sets of coordinated scanners that collectively gain some knowledge about a target network. The result is a description of scan campaigns that summarize the holistic scan activity of an attacker.

4.2.1 Attacker Model for Distributed Scan Campaigns

To start a network communication, the initiator of a flow (cf. Section 2.1.1) reaches out to a service that is known to run behind an open port on the destination host. This eventually leads to a successful establishment of the communication and a series of packets. However, congestion in the network or downtime of the service can lead to failed establishments without any malicious intention of the initiator. However, the intention is different and considered malicious if it is unknown to the initiator whether a service usually runs on a specific IP and port tuple. Such a communication attempt to a specific IP and port tuple to verify the port status is denoted as a *scan probe*. An established flow proves an open port to the initiator and, therefore, a running service. However, most of the guessed ports are probably closed, resulting in failed establishments. A malicious source IP that sends scan probes is denoted as a scanning node, i.e., *scanner*. Furthermore, the term *port scan* describes all scan probes of a specific scanner. An IDS usually reports such failed communication attempts as port scan alerts when exceeding the threshold for a single source IP.

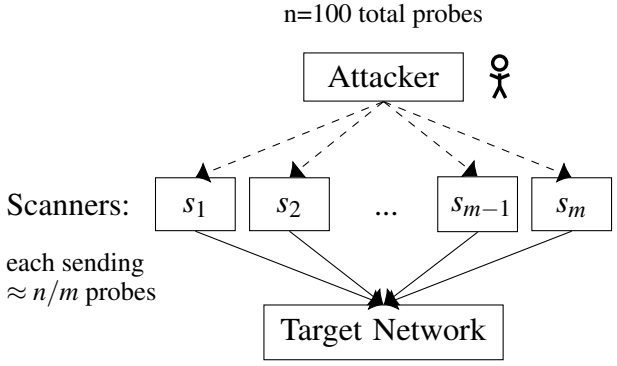


Figure 4.7: Schema of a distributed scan campaign, in which the attacker controls scanners (dotted lines) to make them scan a target network (solid lines).

Typically, many ports and hosts are scanned consecutively by an attacker in a *scan campaign* to gain knowledge about a target network. An attacker, whose scan activity originates from a single scanner, can be detected because this source attempts to establish an extraordinary amount of network flows or because many network flows from this source are not established successfully. To camouflage the scan target or to evade detection in the first place, the attacker can leverage multiple machines that work together as *distributed scanners*. In this scenario, each scanner only originates a fraction of the overall scan activity (cf. Figure 4.7). Their individually gained knowledge from the scans is collectively assembled and reflects the intended target of the campaign.

The remaining of this section describes a two-step approach to detect related scan activity from distributed scanners that are part of a larger scan campaign.

4.2.2 Characterizing Port Scans

When performing distributed campaigns, the attackers can leverage distributed scanners for scanning the target network. The attacker might decide to speedup the scan when using the full scanning resources in parallel. Alternatively, the attacker decides to stay undetected by coordinating the scan activity in a distributed and stealthy fashion [Dai+15]. The two perspectives of characterizing a large network scan are: (1) the information gained about the target and (2) the techniques used to retrieve the information. These two perspectives are reflected by characteristics about the attacker and the target of the scan campaign. In general, distributed scanners are assumed to use that same technical scanning tools and therefore to show the same scanning behavior.

The following presents ten key features that characterize a port scan from the point of view of either the attacker or the target. Intuitively, similarity of these features among scans indicate a relation among them. For large sets of scan activity, these features are evaluated for each scanner. They are summarized in Table 4.2 and utilized in the second step of the algorithm to correlate port scans (cf. Section 4.2.3). The usefulness of these ten features regarding characterizing ports scans and identifying distributed scanners will be evaluated in Section 6.4.1.

Feature	Attacker	Target	Values
Source Ports	X		$[\mathscr{S}+p,\ \mathscr{F},\ \mathscr{M}]$
Destination Ports		X	$[\mathscr{S}+p,\ \mathscr{F},\ \mathscr{M}]$
Vertical Scan		X	$[\text{true},\ \text{false}+h)]$
Horizontal Scan		X	$[\text{true},\ \text{false}]$
Scan Validation	X		$[\text{true},\ \text{false}]$
IP Version		X	$[v4,\ v6]$
Target Hosts		X	$n \in \mathbf{N}$
Scans Probes		X	$n \in \mathbf{N}$
Source Subnet	X		h
Source Location	X		coordinates (x,y)

Table 4.2: Key features to characterize scans regarding the attacker and the target. Values are absolute numbers (**N**) or in categories of Single ($\mathscr{S}$), Few ($\mathscr{F}$), Many ($\mathscr{M}$) with an optional concrete port number p or host IP h.

Source and Destination Ports The involved ports during a scan are of interest because of two reasons. First, the destination port is directly related to the target of the scan. Second, the usage of the source port across port scans contributes to fingerprinting the attacker. Instead of only looking at actual port values P, source and destination ports are also assigned a category depending on the number of different ports. The label *Single* ($\mathscr{S}$) stands for a single port, *Few* ($\mathscr{F}$) abstracts the count of different ports to be in the range of $[2;X]$, and *Many* ($\mathscr{M}$) labels the occurrence of more than X different ports. The threshold X to assign a scan with more than one port the label either $\mathscr{F}$ or $\mathscr{M}$ is subject to evaluation.

$$sim_{ports}(P_1,P_2) = \begin{cases} 1 & \text{if } label(P_1) \cap label(P_2) \in \{\mathscr{F},\mathscr{M}\} \\ 1 & \text{if } label(P_1) \cap label(P_2) = \mathscr{S} \wedge value(P_1) = value(P_2) \\ 0.5 & \text{if } label(P_1) \cap label(P_2) = \mathscr{S} \wedge value(P_1) \neq value(P_2) \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

The similarity between source ports and destination ports sim_{ports} is 0 when the categories mismatch and 1 when the categories *Few* or *Many* match. For the *Single* port category, also the port number p is required to match, otherwise the similarity is only 0.5. This is former formalized for two port sets P_1 and P_2 that are checked with sim_{ports} in Equation 4.1.

Vertical and Horizontal Scans An indirect characteristic of the scan target can be obtained by observing the number of different scanned hosts and the number of different scanned ports. If a scan encompasses more than one target host, it is classified as vertical. If a scan encompasses more than one port on the same target host, it is classified as horizontal.

The similarity between two scans is 1 if both are labeled vertical or horizontal, respectively. If they are labeled differently, the similarity is 0. When the scan targets a single host, i.e., not vertical, the probes of both scanners are additionally enforced to target the same host h.

Scan Validation Not only legitimate network flows sometimes break or fail because of technical reasons, but scans eventually fail in detecting a running service, although the respective port is open. To compensate false negative scan results because of packet loss during the scan, scanners might try several attempts for the same host and port. Thus, scanners are classified according to validating their scan results when they perform more than one scan attempt per host and port. The same classifications for two scans lead to a similarity of 1 and is 0 otherwise.

IP Version In practice, the IP protocol versions 4 and 6 are used. When the versions match, the similarity is set to 1, and to 0 otherwise.

Magnitude of Target Hosts and Scans Probes Especially in well-coordinated scan campaigns, the scan activity is equally distributed among powerful scanner machines. When each scanner contributes equally to the knowledge gain, they scan the same amount of hosts and ports, respectively. For statistical and operational reasons, it is unlikely to see the scan activity being perfectly distributed among the scanners. In fact, the number of probes are expected to differ

among the distributed scanners. The challenge in comparing these numbers is to decide if two scanners still have roughly the same number of if the difference is to high.

$$sim_{magnitude}(a,b) = \begin{cases} 1 & \text{if } |a-b| < min(a,b) \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

Using the absolute number $n \in \mathbf{N}$ of probes and their difference for two scanners is not suitable, because the absolute difference means different things for probe counts in the magnitude of hundreds or thousands. Thus, the similarity of scanned hosts and ports between two port scans is taking the order of magnitude into account. Empirical tests indicate that the following calculation meets the concerns: The difference between two numbers a, b must be smaller than the smaller of both numbers. In this case, the similarity is 1 and 0 otherwise (cf. Equation 4.2).

Source IP Subnet and Geolocation Another indicator for distributed scanners is the proximity between them. This is eventually with respect to their topological or geographic location. Although this does not apply when utilizing a botnet with infected machines around the globe, it is likely to apply when using the servers of a specific provider or data center. For that, the source IP of the scanner and the coordinates of its geolocation are leveraged.

Ideally, two scanner IPs would be checked if they belong to the same subnet, if known. Instead, a more generic calculation is applied with a linear similarity between 0 and 1 depending on the number of equal leading bits in the IP. If two IPs have the same prefix of length 27 bits, their similarity is $27/32 = 0.84$. For the geolocation, countries and coordinates are distinguished. If coordinates only differ in a few degrees, the similarity is 1. The similarity is 0.5 if the coordinates are still in the same country and 0 for different countries.

4.2.3 Correlating Port Scans

The following describes the second algorithm part to correlate scanners that are coordinated in a scan campaign. The correlation algorithm leverages the characterization of port scans according to the ten key features from Section 4.2.2 that fingerprint scanners and their scans regarding both attacker and target.

Before the correlation is performed, a sanitization of the characterized scans filters false positives that are likely caused by other technical reasons like network issues that happen from time to time. In contrast, malicious port scans probe many hosts and ports to gain the desired knowledge, which causes many failed communications, e.g., broken TCP flows. Thus, the threshold ε requires a minimum number or scan probes in a port scan. If less, the respective suspicious IP is filtered and not considered for the correlation of port scans.

Two scanners are believed to collude if the similarity of their fingerprints exceeds a threshold t. The similarity over the ten key features is defined as a weighted average:

$$sim = \frac{\sum_{i=1}^{10} s(i) * w(i)}{\sum_{i=1}^{10} w(i)}$$

This similarity function compares each of the ten features pair-wise for two port scans. Each feature similarity $s(i)$ for the features $i \in [1;10]$ is in the range between 0 and 1. Features can

be assigned weights $w(i) \in [0;1], \sum w(i) = 1$ when calculating the similarity among port scans that is $0 \leq \text{sim} \leq 1$. These weights prioritize certain features that indicate distributed scanners with higher certainty. The threshold t controls if two scans are similar enough regarding their attacker and target to be assumed to belong to the same campaign.

After the similarity between any two per scans is calculated using *sim()*, hierarchical clustering finds distributed scanners in the large scan alert set. This type of clustering benefits from its parametrization as no prediction about the number of clusters or their sizes is required. Instead, the threshold t is incorporated to allow an interactive inspection of the resulting clusters when varying this parameter. In particular, the clustering algorithm *unweighted pair group method with arithmetic mean* (UPGMA [Joh67]) is chosen for two reasons. First, the bottom-up property of this clustering algorithm ensures that the most similar scans become clustered first. Second, UPGMA recalculates similarities of merged clusters by averaging the similarities of all contained elements. As most of the key features have only a few possible values (cf. Table 4.2), this tries to preserve the clusters dominating features when searching for new elements to merge.

After characterizing the scan activity of each scanner and correlating scanners based on their characteristics' similarity, the result is a campaign description that identifies related scanners and their scan activity. Aggregating the scan activity of distributed scanners allows the victim to reconstruct the full scope of the scan campaign. Besides that, the scan description highlights the campaign's characteristics when looking at the most similar features among the distributed scanners.

4.2.4 Summary of Scan Campaign Detection

A detection algorithm has been presented to detect port scan campaigns together with all the related scan activity, even when the activity originates from different sources on the Internet. The algorithm makes use of the combination of activities that together show a common behavior, both regarding the attacker and target. While assessing the scan activities individually potentially indicates the ongoing port scan, only assessing the homogeneous scan activities collectively reveals the bigger picture of the scan campaign. Detecting campaigns with this algorithm benefits from a better accuracy because the reduction of false positives increases the precision. Furthermore, the right priority can be assigned to scans that seem to be small and unrelated but would usually be ignored instead of being seen in the context of the larger campaign.

The algorithm for the detection of scan campaigns is working in two steps. It first characterizes the scan activity for each source individually. The resulting fingerprint consists out of ten key features that characterize the scan activity regarding the attacker and target. The second step of the algorithm leverages these characteristics to link similar fingerprints. Their aggregated scan activity and its fingerprint then highlights the campaign's characteristics. The parameterization of the ten key features and their prioritization is investigated in Section 6.4.1.2.

The more activity of the campaign is monitored, the clearer becomes the picture of the campaign. A good detection accuracy, therefore, requires to capture a large portion of the relevant network traffic and the scan activity in particular. Besides the application of the detection algorithm in stub networks, their collaboration or the application in a transit network, i.e., at an Internet service provider (ISP), would increase the detection accuracy of large scan campaigns with Internet-wide scope. Section 6.4.1.3 investigates to which extent scan campaigns can be detected without having a global monitoring of the Internet.

4.3 P2P Botnet Detection

Botnets are a major threat to any Internet user and the Internet infrastructure itself. This is not only because infected hosts can harm the individual users, e.g., by stealing their banking credentials [And+13; OBr16]. When collectively controlled by the botmaster, the infected hosts, i.e., bots, can also bundle their resources to perform click fraud [Wyk12] or distributed denial-of-service (DDoS) [Sch+10]. But most important, a bot running in a network potentially poses a foothold for the attacker in it. To effectively stop a botnet, it is necessary to identify the bots in the first place. Especially P2P botnets are hard to take down, as bots disseminate commands among each other and there is no central command and control (C2).

In contrast to a traditional detection on the basis of individual network flows, the characteristic P2P communication among the bots can be leveraged to detect a botnet itself, including its communicating bots. The detection, however, requires to capture the communication behavior of many bots. While monitoring a local network reveals the Internet communication with remote hosts, potential communication among the remote hosts is impossible to capture for the local monitoring. Thus, a local measure would never be able to identify the characteristics of a P2P botnet. Instead, large-scale monitoring data is required to correlate traffic from different network sites. This would finally allow to detect communication characteristics among hosts that equal those of a P2P botnet.

Based on statistical algorithms that operate on NetFlow data to identify infected machines in large networks, this section proposes a detection algorithm based on the statistical concept of random walks. Other algorithms presume visibility on large parts of the communication within the botnet. In addition, so far, only structured P2P botnets have been addressed, while especially more recent botnets shifted to unstructured P2P approaches. For their detection, this section revisits and simplifies the BotGrep algorithm in [Nag+10] (cf. Section 3.4.1). The modified version makes use of the less complex clustering algorithm density-based spatial clustering of applications with noise (DBSCAN) instead of the SybilInfer algorithm [DM09]. In addition, post-processing has been removed to reduce computational overhead.

This section revises parts of the conference paper [Muh+18] that resulted from supervising the first author during his internship. The section first introduces a formal model that describes the communication of P2P botnets and how this is related to benign Internet communication. Afterward, different forms of network visibility are introduced and how they affect the monitored botnet communication. Last, this section discusses the detection of bots in large communication graphs using random walks.

4.3.1 Detection Model on Communication Graphs

The communication graph used for the botnet detection represents the communication in a network, i.e., on the Internet, on the basis of hosts. The *real-world communication graph* $G_N = (V_N, E_N)$ consists of vertices V_N representing the hosts in the network. The directed edges E_N among the vertices in the graph reflect who is talking to whom in the network. In practice, this means that only NetFlow data is required for the proposed detection of P2P botnets.

This graph G_N contains all communication among hosts, including both legitimate flows and those that stem from botnet communication. Note that multiple network flows among two hosts will still end up in a single edge in the graph between the respective two vertices. More

formally, the *legitimate communication* in the network is modeled by the graph $G_L = (V_L, E_L)$, whereas the *botnet communication* is modeled by the graph $G_B = (V_B, E_B)$. Merging both graphs results in the overall communication graph, i.e., the set of hosts $V_L + V_B = V_N$ and the set of communication relations $E_L + E_B = E_N$. However, neither V_L and V_B nor E_L and E_B need to be disjoint. In fact, it is likely that communicating bots V_B also emit legitimate traffic and, therefore, are also included in V_L, i.e., $V_L \cap V_B \subset V_N$. Consequently, almost every host is expected to emit legitimate traffic, i.e., $V_L \approx V_N$, whereas the number of bots is only a small portion of the overall hosts, i.e., $V_L >> V_B$.

The goal of the botnet detection is to analyze the communication graph G_N and to extract those nodes that are part of the botnet, i.e., $b \in V_B$. Although the detection is based on the characteristic botnet communication, i.e., the edges E_B in the botnet graph G_B, communication relations among the bots are not intended to be included in the detection result.

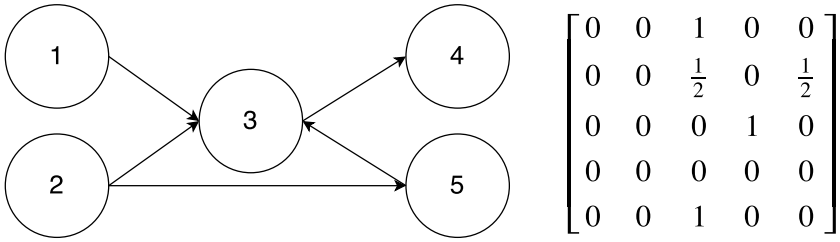


Figure 4.8: A directed graph and its corresponding adjacency matrix with probabilities.

For an efficient implementation of the graph, the concept of an adjacency matrix as a two-dimensional array of size $|V_N| \times |V_N|$ is used. Usually, the possible values $0, 1$ indicate the presence of an edge only. For an efficient applicability to random walks, the definition of the adjacency matrix is slightly adapted here. In particular, the values $[0; 1]$ in the adjacency matrix not only indicate the presence of an edge, i.e., when the value is greater than 0, but also assign the edges a weight, i.e., the transition probability to another node. This follows the idea of BotGrep (cf. Section 3.4.1) and is furthermore illustrated in Figure 4.8. The matrix should be read as follows. Each row $v_i \in V_N$ reflects a specific vertex with its outgoing edges to any other vertex $v_1, v_2, \ldots, v_{|V_N|}$. Thus, for example the third value in the first row determines the probability of jumping from vertex v_1 to vertex v_3. The sum of all probabilities in every row must equal 1. When initially creating the adjacency matrix for a given graph, the probabilities of outgoing edges are equally distributed, as formalized in Equation 3.1 and visualized as example in Figure 4.8. While calculating random walks for the detection, the distribution of probabilities changes. The details of updating the adjacency matrix is detailed later in Section 4.3.3. In the final updated matrix, accumulating all path probabilities from v_i to v_j results in the random walk's final probability to end at v_j after a fixed number of hops starting from v_i. This probability of a node to be the end-node of a random walked is then leveraged for the P2P botnet detection in Section 4.3.3. However, first the restrictions of communication graphs in practice are highlighted.

4.3.2 Restrictions on Communication Graphs

The communication model in Section 4.3.1 captures the ground truth of who is talking to whom with $G_N = (V_N, E_N)$. However, in practice, NetFlows as basis for communication graphs are likely to not exactly match the ground truth, which results in the monitoring graph $G'_N = (V'_N, E'_N)$,

as detailed in the following two paragraphs. They describe how the restrictions affect the communication model.

NetFlow Restriction The first reason for restrictions is caused by an inaccurate monitoring of communication relations. This means that not all communicating host pairs are identified, which results in some edges in G_N to be missing. Depending on the monitoring in place, this is caused by sampling on either packet or flow level. Either way, the sampling can be unintentionally because of unexpected overload situations or intentionally for performance reasons [Est+04]

Note that in practice not every missing packet or flow in monitoring directly causes a missing edge in the communication graph. This is because all packets of a flow would have to be missed before the whole flow is missing. Similarly, all flows between two hosts would have to be missed before the edge in the graph is missing. However, a host usually establishes several flows to a destination, e.g., parallel connections for HTTP or both control and data connection for File Transfer Protocol (FTP). For simplicity of the communication model, however, the NetFlow restriction effectively reflects the monitoring to miss the communication between two hosts altogether. Furthermore, if all the communication of a host is missed, this node is not included in V_N. More formally, this results in a subset of edges

$$E'_N \subseteq E_N \quad \text{with } |E'_N| = k \times |E_N|,$$

where $k \in [0,1]$ is the fraction of contained communication relations. The subset of vertices is then determined by

$$V'_N = \{v_i \in V_N \quad | \, \exists v_j \in V_N : \{(i,j),(j,i)\} \cap E'_N \neq \emptyset\}.$$

Visibility Restriction The second reason for restrictions is caused by the supervised system itself, i.e., the monitored network. The monitoring system can only capture traffic that is transmitted through the network, but cannot capture global. Internet traffic of other networks. This is referred to as visibility. The result of the visibility restriction is that only the communication of particular hosts is monitored, including any incoming and outgoing flows with any remote host. More formally, the visibility restriction is based on an initial pick of hosts $V_I \subseteq V_N$ with $|V_I| = k \times |V_N|$ where $k \in [0,1]$ is the fraction of contained hosts. Starting with these initial host, the restriction leads to a subset of edges that is

$$E'_N = \{(i,j) \in E_N \quad | \, \{v_j, v_j\} \cap V_I \neq \emptyset\}.$$

Based on the monitored communication relations, the subset of vertices in the restricted graph is

$$V'_N = \{v_i \in V_N \quad | \, \exists v_j \in V_I : \{(i,j),(j,i)\} \cap E_N \neq \emptyset\}.$$

The botnet detection with random walks has to be robust against both NetFlow and visibility restrictions, and is subject to evaluation. The detection must work also in these cases when the restrictions in question are in place, which cause the detection to work on G'_N instead of the global view G_N. In addition, the detection must also be resilient against the restrictions so that it is still able to detect the botnet and a majority of its bots.

4.3.3 Botnet Detection with Random Walks

The botnet detection operates on the communication graph G'_N. At the beginning of the detection, the adjacency matrix is initialized for G'_N with the probabilities described in Section 4.3.1. The following two paragraphs describe the steps to identify the bots of a P2P botnet in the communication graph, represented by the adjacency matrix P_{ij}.

Random Walks A random walk starts at round $r = 0$ at a random row $i_{r=0}$ of P_{ij} and uniformly picks one non-zero element $j_{r=0}$ in this row. This is equivalent to starting at a random node and choosing a random edge to continue the walk. When starting, the algorithm must not pick a row that contains only zeros, because this means that the respective node is isolated and there are no edges to continue the walk. The walk continues at round $r = 1$ when the destination of the chosen edge becomes the new start of the next walking step, i.e., $i_{r=1} = j_{r=0}$. In total, this is repeated k times, so k represents the length of a random walk.

The botmaster benefits from the fact that bots in the P2P botnet are well-interconnected. Not only obviously because the bots cannot simply be paralyzed by law enforcement taking down the centralized C2 infrastructure. Especially the high density of interconnections has additional benefits. The botmaster can take cover when submitting commands to a single bot only and letting the botnet itself to propagate the command to the remaining bots. With higher density, the command propagates faster. A high density also makes individual bots robust against losing connection to the botnets when other bots turn off or get cleaned from malware. Because of these benefits, bots in V_B are expected to be well-interconnected among each other. The dense structure will be reflected both in the botnet graph G_B and the monitored graph G_N. Consequently, a short random walk starting from a node $v_p \in V_B$ will have a higher probability of ending in another node $v_r \in V_B$ than a walk starting from an ordinary host $v_l \in V_L$ if $v_l \notin V_B$.

Due to this so-called *fast-mixing* property of the P2P subgraph, its state probability mass is closer to the stationary probability distribution than the infected slow-mixing rest of the network [Sin92]. The effect becomes apparent when performing a large number of random walks on the graph. After n random walks, each of length k, the walking algorithm terminates. Then, the probabilities of all walks are accumulated to determine the likelihood of each vertex to be an end-node. As the authors of [Nag+10] point out, well-connected nodes with high hub and authority ranks speed up the mixing rate. This is an unwanted effect as legitimate hubs and authorities might attract more random walks than the well-connected botnet graph G_B. However, it can be assumed that a malicious subgraph, i.e., the botnet, has more hubs and authorities due to its P2P topology. That is why, similarly, a dampening constant can be applied to suppress the effect of individual well-connected nodes and thus normalize the probability distribution to enable clustering.

The result of this first step with random walks is a vector of size $|V_N|$ that holds the normalized probability for every host to be the end-node of a random walk.

Clustering The list of hosts and their respective probabilities now has to be filtered for hosts with normalized probabilities that are similar to each other. Nodes of a P2P botnet will have similar probabilities. Therefore, the density-based clustering algorithm DBSCAN is used to cluster the resulting probability distribution with respect to the distances among the data points. DBSCAN requires two parameters: (1) ε denoting the neighbor range for each node and (2)

minPts as the minimum number of points a cluster should have. To avoid many small clusters, preliminary experiments indicate that the minimum size should be set to 50, while $\varepsilon = 0.6$. An advantage of DBSCAN is that it does not require the number of clusters in the data a priori, unlike the k-means algorithm. Furthermore, DBSCAN is able to take outliers and noise into account.

The normalized random walk state probability mass of the set of bots V_B is homogeneous and should be different from the slow-mixing rest, i.e., the uninfected hosts in the communication graph $V_U = G_N \setminus G_B$. After clustering, the final result is the separation of hosts V_N in the network communication graph G_N into hosts V_B that seem to participate in a P2P botnet and into hosts V_U that communicate legitimately only. Afterwards, this would allow to extract the respective botnet subgraph G_B and leave the legitimate communication graph G_L.

4.3.4 Summary of Botnet Detection

To identify communicating bots within the large amount of network communication every day, a detection algorithm has been presented that identifies the characteristic communication behavior of an unstructured P2P botnet among the bots. Based on the related work of Nagaraja et al. for structured P2P botnets [Nag+10], the algorithm makes use of the combination of bot activities that together exhibit this botnet-specific characteristics. Assessing the bots' activities individually would probably fail as long as no IDS signature of the particular botnet exists. In contrast, the presented algorithm assesses the homogeneous bot activities collectively to enable its detection in the first place and to reveal the bigger picture of the whole P2P botnet. Another advantage of the detection algorithm is that the alerts for communicating bots are directly related to each other. If the alerts for each bot would be reported without any relations, assembling the bot alerts to a single botnet is an additional effort for the human operators. Instead, the output of the presented algorithm already indicates the alerts that belong together.

The algorithm for the detection of unstructured P2P botnets is based on the key insight that a regular communication graph on the Internet statistically differs from the communication graph of bots within the P2P botnet. This fact stems from the P2P nature of these botnets that results in a high density among the bots regarding their communication relations. Using a random walk approach, the detection algorithm separates nodes in the monitored communication graph into nodes that collectively show P2P characteristics and nodes that do not.

The more activity of the botnet is monitored, the clearer becomes the P2P characteristics among the bots. A good detection accuracy, therefore, requires to capture a large portion of the relevant network traffic and the bots' activity in particular. However, several restrictions regarding the network visibility have been presented that exist in practice, including that an Internet-wide capture required to encompass all bots is impossible. Because of these restrictions, the detection algorithm must not rely on full network visibility but must be resilient against incomplete communication graphs. The experiments in Section 6.4.2 evaluate to which extent of restricted visibility a significant statistical differences between the communication graphs is apparent.

4.4 Summary of Security Monitoring

Security monitoring with a NIDS is challenged by restricted visibility, which causes negative effects and lead to a decreased detection accuracy. The approaches presented in this chapter mitigate these effects and lead to a better visibility on the attack. Their deployment is not exclusive but should be combined along the intrusion detection process (cf. Section 3.2.1) to be most effective. The first dimension to extend the visibility correlates host and network data already during monitoring in real-time. In the second dimension, detection algorithms leverage correlated monitoring data using the example of network data to identify larger attack scenarios in the communication graphs. In their combination, the presented approaches extend the visibility of an IDS to enable better detection accuracy

The foundation of a good detection accuracy is high-quality monitoring data. Without this, no sufficient protection, especially against sophisticated attacks, i.e., advanced persistent threats (APTs), can be provided in the first place. To achieve a better monitoring visibility, the *zeek-osquery* platform has been presented in Section 4.1. It ties activities of the host to those in the network and allows for the analysis of the resulting data in real-time. This integrated visibility on the actions of the attacker sheds light on their causal connection. For that, zeek-osquery combines the network security monitor *Zeek* with the host sensor *osquery*. A new framework extends Zeek by the scalable orchestration of osquery sensors, interactive queries for host data, and the analysis of this data directly in Zeek. This way, the enriched monitoring data by host context not only assists forensic analysis through more detailed log data but also live intrusion detection. The basis of the real-time analysis is the attribution of network flows to their host processes. While necessary host data can be retrieved efficiently in a continuous stream of events, additional data can be queried interactively on demand. This enables a privacy-respecting retrieval of sensitive host data because it is dynamically retrieved for certain reasons only. This particular use-case is demonstrated by the hosts providing cryptographic key material to assist in the inspection of TLS connections. The zeek-osquery platform is evaluated in Section 6.3 regarding the accuracy of network attribution, performance, and its usability for intrusion detection.

The fine-grained correlation of host and network data, however, requires administrative access to the hosts for deploying the osquery host sensors. Although suited for certain environments, this is not always possible. Thus, this section furthermore pursues the hardening of detection algorithms to make them more resilient against restricted network visibility. Thus, this chapter presented two scenario-specific detection algorithms that are based on the same common principle. The basic idea for increasing the detection accuracy is to exploit that some scenarios cause a lot of malicious activity. Assessing a single or few of these activities independently from the others might not give enough evidence to detect the whole scenario. Instead, when assessing a significant amount of the malicious activity together, the characteristics of the scenario become clearer and give more certainty in the detection result or enable the detection in the first place. This principle is applicable, especially for distributed attack scenarios that cause a lot of similar activity. In contrast to detection algorithms that assess events individually and potentially report a sheer flood of alerts, the presented algorithms already identify that the malicious activities belong to the same attack.

In particular, this chapter presented detection algorithms that leverage correlated network data, i.e., the communication graph, for the detection of scan campaigns with coordinated scanners (cf. Section 4.2) and P2P botnets (cf. Section 4.3). These detection algorithms are supposed to

detect potentially global attacks with their local network visibility only. In the case of the scan campaigns, the characteristics of similar scanning behavior among the coordinated scanners enables the detection to be robustness against restricted visibility. The scan activity of every source is first characterized. In the aftermath, sources with similar characteristics are identified. When coordinated scanners have been identified, their scan activity is aggregated. In the best case, the aggregation reflects the whole scan campaign. But even when not all scan activity or not every coordinated scanner is captured by the monitoring, the activity that is captured can still be assembled to detect coordinated scanners and partly represent the scan campaign. In the case of P2P botnets, their robust detection against restricted visibility is enabled by the characteristic communication channels that bots establishing among each other. This communication behavior is statistically different from regular communication behavior on the Internet like it is seen with the common client-server pattern. From the point of view of an individual bot, one can only guess about ongoing P2P communication. However, the view on a collection of bots reveals their inter-connections. As the bots' communication behavior differs significantly from others, it is still apparent even when not all bot communication is captured. Both, the detection of scan campaigns as well as the detection of P2P botnets, are evaluated regarding their accuracy in Sections 6.4.1 and 6.4.2, respectively.

The next chapter continues with those approaches to an IDS that correlate alert information for a better picture of the whole attack.

5 Correlation of Network Alerts

The outcome of security monitoring reflects malicious activity that an IDS reports as alerts. An isolated view on these low-level IDS alerts, however, misses the context of the attack they belong to. The resulting incomplete view on attacks often renders an effective mitigation of the attack impossible. Thus, as part of the intrusion detection process as described in Section 3.2.1, alerts need to be summarized and correlated to obtain the bigger picture of an attack.

However, there are two particular challenges for reconstructing network-wide attacks from their alerts. First, the large alert volume overwhelms security operators when tying to identify relations among the alerts. Especially the big picture of distributed attacks can rarely get assembled when the relations of alerts from coordinated sources stay hidden from the security operators. Second, stealthy and comprehensive attacks, i.e., advanced persistent threats (APTs), additionally impede their reconstruction because of spatially and temporally distributed alerts. As a result, such stealthy attacks result in only a few alerts, while at the same time a large number of alert for conventional attacks are reported. Failing to link these alerts can result in the attacker to succeed without being noticed for a long time.

The alert correlation – the automated assembling of alerts to a descriptive summary of attacks – therefore, has to overcome the challenges of temporally and spatially dispersed alerts. Thus, the alert correlation algorithms presented in this chapter leverage different kinds of similarity among alerts and attacks, respectively. The correlation outcome provides the security operators with the attacks to their whole extend without analyzing every single alert themselves.

In particular, this chapter first defines and summarizes three stages of the alert correlation process in Section 5.1 for the processing of IDS alerts into attack representations. The subsequent two sections of this chapter present concrete correlation algorithms that implement these process stages. Algorithms of the first stage in Section 5.2 group all alerts that directly belong to the same attack or attack step. Section 5.3 presents algorithms that combine the second and third stage to link attacks with each other based on additional information about the attack scenario. More precisely, this chapter presents an aligned solution denoted as *graph-based alert correlation (GAC)* for clustering alerts from distributed attacks (cf. Section 5.2.1) and linking alert clusters from multi-step attacks (cf. Section 5.3.1). In addition, the *weak alert correlation* complements the clustering particularly with respect to APT alerts (cf. Section 5.2.2) and *collaborative attack correlation* assists in linking attacks detected in a distributed IDS deployment (cf. Section 5.3.2).

5.1 Alert Correlation Process for Attack Detection

Alert correlation algorithms ease the task of analysis alerts by correlating alerts with each other to obtain the bigger picture of an attack. A common approach is to group alerts based on similar attributes [ZLK09; Jul03; Vas+15a; Loc+05] like source and destination IP, and to report common attribute patterns (cf. Section 3.4.2). Other approaches correlate alerts of multi-step

attacks [NCR02; Sun+16] to identify and link the individual steps of an attacker, e.g., a port scan that is followed by an exploit of a specific vulnerability on the target host (cf. Section 3.6). Thus, alert correlation is the overall process to make relations among alerts visible, which can serve as indication of them belonging to a larger attack.

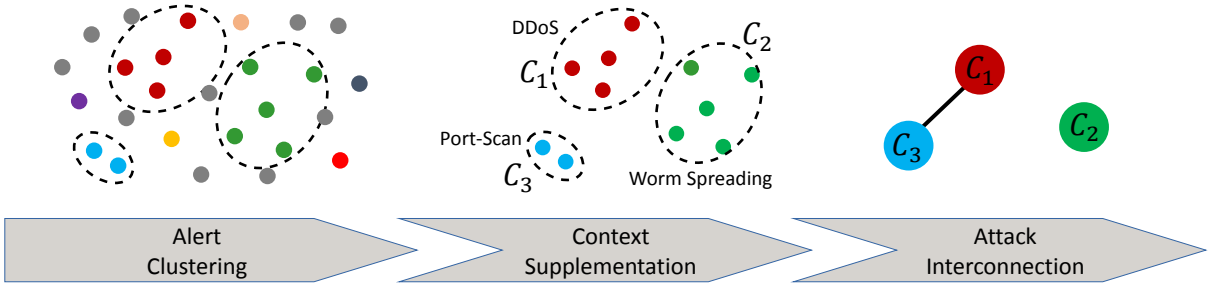


Figure 5.1: General alert correlation process with example for scenario labeling.

The alert correlation process converts a set of alerts into a representation of attacks. This correlation is an essential part of intrusion detection. However, alert correlation does not include intrusion detection and the reporting of alerts. Instead, alert correlation algorithms rely on sensors to classify malicious events as alerts (cf. the intrusion detection process in Section 3.2.1).

This section gives a unified description of the alert correlation process for transparent analysis and comparison of correlation algorithms. It is a revised version of the alert correlation process in the journal article [HF19]. The description characterizes alert correlation algorithms by breaking them down into their core building blocks (cf. Figure 5.1): *alert clustering*, *context supplementation*, and *attack interconnection*. Depending on the goal of specific alert correlation algorithms, individual blocks are less important or not necessary at all. If a step is not addressed, the input of a block is also its output. The following first introduces a formal model for the alert correlation process and then describes each building block in more detail.

IDS Network Alert An alert $a \in A$ can come from arbitrary sources like a network- or host-based IDS and indicates a potential security breach or generic malicious activity in the network. It can be either a true positive or a false positive, depending on the performance of the underlying intrusion detection. Every alert $a \in A$ consists of a fixed vector of attributes $a = (a^1, a^2, \ldots a^n)$, e.g., source and destination addresses, as well as source and destination ports. In case of network intrusion detection, the classification of malicious events as alerts is based on the network communication between hosts (cf. Chapter 4). Any network alert has a randomly assigned unique identifier *(UID)*, the timestamp *ts* of the network flow, the source and destination *IP addresses* and *ports* as well as the transport *protocol*. In addition to the event's attributes, the alert contains the *alert_type*:

$$a := (uid,\ ts,\ src\_ip,\ src\_prt,\ dst\_ip,\ dst\_prt,\ proto,\ alert\_type)$$

IDS Meta Alert An attack, or more precisely all alerts that are caused by the same malicious action, i.e., an attack i, will result in a set of true positive alerts S_i. The set of all alert sets from different steps is given by $\hat{S} = \{S_0, S_1, \cdots, S_{n-1}\}$. Applying alert correlation to this set reduces the alert volume and results in *meta alerts* that abstract and reference a set of at least one IDS alert. For that, a correlation function $\mathscr{K}$ clusters an alert set A into a set of clusters $\hat{C}$, with

each cluster $C_i \in \hat{C}$ being supposed to consist out of all alerts that represent a particular attack step. In contrast to an IDS network alert, a meta alert has its own assigned *UID*, the *time span* between first to last alert, the set of all *alert ids*, the set of attacker and victim *IP addresses* among all alerts, and a *message* that describes the attack. Thus, an IDS meta alert is the set of alerts triggered by the same attack action. It is the outcome of an alert correlation algorithm $\mathscr{K}$ that transforms an alert set A into clusters $C_i \in \hat{C}$:

$$m := (uid,\ ts,\ alert\_ids,\ attackers,\ victims,\ message)$$

A multi-step attack $M_j \subseteq \hat{S}$ contains several single-step attacks. The set of all multi-step attacks is $\hat{M} = \{M_0, M_1, \cdots, M_{m-1}\}$. The following paragraphs describe the tasks in the intrusion detection process as building blocks to achieve a clustering of alerts into clusters $\hat{C}$ and summarizing of these into multi-step attacks $\hat{M}$. Table 5.2 summarizes the notation that is used throughout this chapter to refer alerts when processed by the alert correlation process.

Correlation Stage				
Alert Clustering		**Attack Interconnection**		
$a_k \in A$	$S_i \in \hat{S}$	$l_i \in L$	$M_j \in \hat{M}$	Ground Truth
	$C_i \in \hat{C}$		$I_j \in \hat{I}$	Correlation Result
Alerts	**Alert Clusters**	**Context Labels**	**Attack Clusters**	
Symbol Name				

Figure 5.2: Notation used throughout the alert correlation process.

Alert Clustering The alert clustering will partition alerts into respective clusters of alerts $\hat{C} = \{C_0, C_1, \cdots C_{n-1}\}$. Each cluster $C_i \in \hat{C}$ is supposed to represent an attack and the alerts that belong to it. The alert clusters are supposed to model the actual attacks $S_i \in \hat{S}$, so in best case this leads to $\hat{S} = \hat{C}$. For that, two tasks will be carried out here: *Alert filtering* identifies duplicates as well as false positives among alerts and *attack isolation* clusters alerts that belong to the same attack.

Alert filtering takes care of filtering false positives, so that in the best case it holds true that:

$$\forall a \in A: \quad \exists S_i \in \hat{S} \wedge a \in S_i \iff \exists C_i \in \hat{C} \wedge a \in C_i$$

Please remember that the definition of *false positive* can depend on the context and the attacks to be detected, respectively. Anyway, this does not require a mapping from alerts to clusters with respect to attack steps. Instead, this task requires alerts to be assigned to a cluster and thus be included as input into the next building block if and only if they are induced by an attack in $\hat{S}$.

Attack isolation requires clustering to assign each alert out of set A to one cluster $C_i \in \hat{C}$. The correlated clusters $\hat{C}$ should reflect the original attack steps $S_i \subseteq A, S_i \in \hat{S}$. This task may vary from traditional clustering that aims for high homogeneity within clusters and high heterogeneity among the clusters. Hence, the challenge of clustering in the field of alert correlation is to find an assignment that reflects the reality, which might not be the optimal solution from a data mining point of view.

Context Supplementation After alerts have been filtered and clustered in the previous step, each alert cluster is supplemented with additional context. Such context could be information from knowledge databases, which provide information on vulnerabilities that might be exploited in a particular attack.

Another type of information could be a description of the attack. Thus, giving the clusters a meaningful label eases human analysis. A popular technique is to summarize alert features by finding common attributes in alerts, e.g., the source IP of the attack that might be present in all alerts related to a particular attack, and to suppress alert attributes that have high variance. Description of clusters such as counting the most frequent attribute values or value combinations is highly valuable for analysts. This might be the reason why most approaches for alert clustering tend to search for alert subsets that result in labels that are easy to understand by humans. Although using the most frequent attribute values works well in practice most of the time, this has some limitations as it imposes constraints on the attacks that can be detected.

For context supplementation, each cluster in $\hat{C}$ is labeled with a label $l_i \in L$ that gives additional information, e.g., environmental context or cluster description.

Attack Interconnection The last building block in the alert correlation process attempts to find relations among the alert clusters in $\hat{C}$. The resulting attack clusters $I_i \in \hat{I}$ reflect the assembling of single attack steps $\hat{S}$ into the multi-step attacks $\hat{M}$. An example is a scan for vulnerable web services in a subnet, followed by an exploit against a server in this subnet. To connect such attack steps, usually a combination of sequential- and causal-based correlation mechanisms is applied.

In worst case, each alert results in an own group during alert clustering. The second step enriches each alert with additional information, e.g., information on the vulnerability that is exploited. Then, most effort of the correlation algorithm lies in this third block of attack interconnection, as the definition of attacks is implemented in the dependencies between attacks.

5.2 Alert Clustering

The first stage in the alert correlation process (cf. Section 5.1) identifies alerts that are all caused by the activities or effects of the same attack step. In particular, two challenges exist in this process stage (cf. Section 1.1). First, there are distributed attack scenarios such as distributed denial-of-service (DDoS), port scans, and worm spreadings. These attacks cause a bulk of similar – almost redundant – alerts for the several affected hosts. Second, the occurrence of alerts from slow and stealthy APT-like attacks is a good deal more infrequent compared to alerts from bulk attacks. Consequently, the alerts' weak relations is likely to get lost in the shuffle. Alert clustering in this section analyses the alerts regarding similar features that indicate relations among the alerts and overcomes the two challenges in particular.

This section combines revised parts of the conference paper [HF18] and of the supervised master thesis [Ort19]. The section presents two alert clustering approaches for two different purposes. The first approach processes chunks of consecutive alerts, i.e., batches, and identifies related alerts within each alert batch. The second approach is an addition to the first one and clusters alerts of stealthy attacks that are likely to be filtered by the first approach.

5.2.1 Graph-based Community Clustering

The first approach for alert clustering represents alerts as nodes in a graph and adds edges between similar alerts. This enables the search for alerts sharing similar attributes and the clustering of alerts within these graphs afterwards. The following first describes the used graph model, then the clustering method.

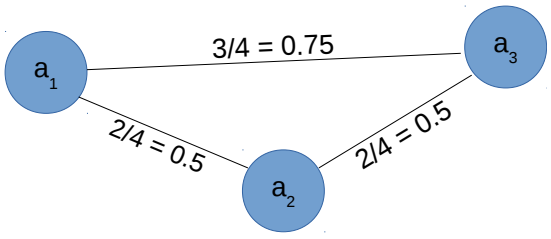
Transforming Alerts into a Graph The set of alerts A is transformed into a weighted *alert similarity graph* $G_{\text{attr}} = (A, E)$ that contains alerts of set A as nodes. Every edge $(a_1, a_2) \in E$ is weighted with the similarity $s = F_{sim}(a_1, a_2) \in [0, 1]$ in between two alerts $a_1, a_2 \in A$. Function F_{sim} compares all n attributes $(a^0, \ldots, a^{n-1})$ of the alerts, respectively, as follows:

$$F_{sim}(a_1, a_2) = \sum_{j=0}^{n-1} c^j \cdot h^j(a_1^j, a_2^j) \tag{5.1}$$

Per attribute, an attribute-specific comparison function h^j delivers a similarity value in $[0, 1]$. All attribute comparisons are weighted according to a vector $c = (c^0, \ldots, c^{n-1})$ such that $\sum c^j = 1$. The edge weight s determines whether a particular edge is present in the graph. The similarity between two alerts is required to be equal or higher than a minimum similarity threshold τ, for an edge (a_1, a_2) to be included in E^τ and G^τ_{attr}, respectively. Thus, τ controls the number of edges $|E|$, by removing edges between alerts that are most probably unrelated. The weight of edges depends on the implementation of F_{sim}. To choose a suitable threshold τ, it is beneficial to know the expected alert similarity among alerts of the attacks that should be detected.

	Source		Destination	
ID	**IP**	**Port**	**IP**	**Port**
a_1	E	X	W	P
a_2	T	Y	W	P
a_3	E	Z	W	P

(a) Example alert set.



(b) The resulting G_{attr} graph.

Figure 5.3: Transformation of an alert set into an alert similarity graph.

With the focus on network alerts, the most important and common attributes that should be supported by any network intrusion detection system (NIDS) and detection method are the four attributes: IP and port of both source and destination. Furthermore, attributes are tested for equal values to keep the correlation algorithm free from additionally required knowledge, such as subnets and the application type related to a port, e.g., Hypertext Transfer Protocol (HTTP) for ports 80 and 443. Thus, according to Equation 5.1, all h^j return 1 for equal attribute values and 0 otherwise. Attribute comparison is weighted equally with $c^j = 1/n$ for any $c^j \in c$. Other examples for attribute selection and h^j can be found in [VS01]. Another method to assign weights is described by so-called log-graph in [Pei+16]. Figure 5.3b shows the *alert similarity graph* with three nodes for the alerts in Figure 5.3a. E.g., $F_{sim}(a_1, a_2) = 0.5$ because two out of four attributes among the alerts a_1 and a_2 are equal.

Clustering Alerts within the Graph It is important to consider that usually several attackers try to break into an IT system at the same time. Therefore, it is very likely that individual attacks target the same host in the network without a connection between the attacks. One attack might aim to scan for SQL injection on a specific webserver, whereas a second unrelated attack scans for webservers in the same subnet. As both attacks hit the webport on the same server, they will probably be linked with a non-zero similarity even in the filtered graph G^{τ}_{attr}. Therefore, alerts of individual attack steps are further separated by clustering them, i.e., identifying subgraphs in G^{τ}_{attr}. The motivation for clustering is illustrated for an example graph G_{attr} in Figure 5.4. In the bottom one of the two isolated components, one can intuitively identify three loosely coupled subgraphs. These clusters (marked with red circles) are no isolated components, because a few of their alerts are similar to alerts of other clusters. These clusters are supposed to reflect individual attack steps. That some similarity exists between the clusters eventually indicates that they are related. Relations among the clusters, however, is out of scope at this processing stage.

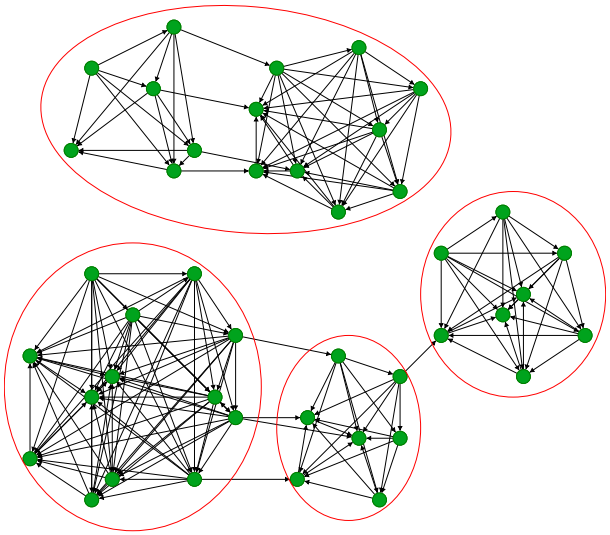


Figure 5.4: Community clustering in an exemplary alert similarity graph.

The alert clustering leverages the graph structure to identify and isolate subgraphs of loosely coupled attacks. For that, community clustering is used, especially the *clique percolation method* (CPM) [Pal+05], to cut the connection between loosely coupled clusters. CPM detects communities by searching for k-cliques, i.e., fully connected subgraphs of size k, that share $k-1$ nodes. This correlates well with the homogeneous inter-connections among alerts in the G_{attr}. These result, e.g., from distributed attacks, which cause several alerts with similar attribute patterns. Furthermore, k-clique communities can also reflect uncertainty in clustering as it allows to assign a node to several communities, i.e. clusters. Although clusters then potentially contain alerts of unrelated attacks as well, they will more likely include all true positive alerts.

This clustering performs well, especially on bulk attacks that cause many alerts roughly at the same time. Infrequent alerts from stealthy attacks, however, are likely to fall into separate alert batches, for which this graph-based alert clustering fails. Such alerts are incorporated into clustering by the following approach.

5.2.2 Weak Alert Correlation

In addition to the bulk attacks detected by the graph-based community clustering of alerts from Section 5.2.1, the following introduces the notion of *weak alerts* that are likely to result from stealthy attacks such as APT attacks. Thus, instead of filtering those alerts as irrelevant that do not assemble to a cluster immediately, the *weak alert correlation* approach can cluster these alerts over long time instead. For that, the approach first groups weak alerts to intermediate aggregations. This data structure enables runtime efficient updates. Furthermore, the aggregation algorithm runs continuously and is not restricted to individual alert batches. Once an aggregation becomes stable, the aggregation result is turned into a *weak meta alert*, similar to an alert cluster.

5.2.2.1 Weak Alert Model

Compared to the number of alerts from bulk attacks, alerts from stealthy attacks are rare. Because of their infrequent occurrence, they seem unrelated to any other alerts, although related alerts in historical data exist. This definition of *weak alerts* is detailed in the following model.

Weak Alert Definition After clustering alerts in an alert set A with a correlation function $\mathscr{K}$, some alerts remain unclustered, i.e., they are not included in any meta alert m_i or belong to any cluster C_i, respectively. Thus, an alert is considered weak, if it cannot be correlated with other alerts and therefore cannot be related to a specific attack. Thus, weakness is a relative descriptor for single alerts, denoted as binary function $\omega : a \in A \mapsto \{0,1\}$:

$$\omega(a) = \begin{cases} 0, & \text{if } \exists C_i \in \mathscr{K}(A) \wedge a \in C_i \\ 1, & \text{otherwise} \end{cases}$$

This results straight into the definition of what the set of weak alerts W is. It is with respect to an alert set A that has the correlated alerts $CA = \cup_{\hat{C}}$, i.e., the union of all clustered alerts among $C_i \in \hat{C}$. The set of all *weak alerts* is defined as the set difference of all IDS alerts minus the correlated alerts:

$$W = A \setminus CA, \text{ with } |W| = |A| - |CA|$$

Not all weak alerts in W necessarily belong to the same stealthy attack. Thus, this model further introduces two characteristics that are used to separate weak alerts from different attacks.

Large IP networks are usually organized in subnets. Reasons for that include to limit broadcast domains, but also to secure subnets of different trust levels, i.e., zones, by measures like firewalls. Especially in APT attacks, lateral movement in the network of an organization is often seen to access data or services that are not directly accessible from the Internet. To reflect this characteristic of respective attacks, the network is assumed to be organized in zones, e.g., the `Internet`, `Intranet`, or `data center`. The direction between two network zones `A`, `B` is denoted with an arrow symbol as $\texttt{A} \rightarrow \texttt{B}$. Consequently, the *alert direction* of an alert a is given by the network zones of its source and destination IPs:

$$Dir(a) := \texttt{A} \rightarrow \texttt{B} \text{ iff. } a.src\_ip \in \texttt{A} \wedge a.dst\_ip \in \texttt{B}$$

Another attack characteristic is how many attackers and targets are involved in the attack. Especially distributed attacks with many involved system potentially cause many alerts. Thus, the relations among attackers and targets and their numbers, i.e., the *attack topology*, characterizes the attack type. The weak alert model distinguishes four topologies:

- One-To-One `(OtO)`: Single attacker and target
- One-To-Many `(OtM)`: Single attacker and multiple targets
- Many-To-One `(MtO)`: Multiple attackers and single target
- Many-To-Many `(MtM)`: Multiple attackers and targets

Similar to the correlation of IDS alerts to meta alerts (cf. Section 5.1), weak alerts are filtered and correlated to reveal their relations to APT attack steps. The outcome are *weak meta alerts* that have the fields *label* for the attack topology, the *alert direction* of two network zones, and the *alert frequency* in addition to a regular IDS meta alert. The attack topology and alert direction characterize the APT attack regarding lateral movement and the alert frequency is an indication for the stealthiness of the attack. Thus, a weak meta alert is the set of weak alerts triggered by a potentially stealthy and long-lasting attack:

$$w_{ma} := (uid,\ ts,\ label,\ direction,\ alert\_ids,\ attackers,\ victims,\ freq)$$

The problem that arises in practice is that the correlation function $\mathscr{K}$ usually runs on a finite alert set A. Thus, $\mathscr{K}$ is only applied on consecutive alert subsets, i.e., batches, reflecting a certain time span. Especially alerts of temporally dispersed attacks probably spread into several alert batches. However, unfavorable circumstances can also cause the isolation of few alerts from bulk attacks to fall into a separate batch, eventually being classified as weak alerts.

Sensitivity and Specificity of Weakness Weak alerts that result from filtering are not necessarily weak when considering multiple batches. Figure 5.5 illustrates five possibilities how weak alerts might relate to other batches. Batches are marked gray, the colored dots depict alerts. Green alerts are successfully correlated, blue alerts are classified as weak. For lanes 1-3, a small portion of alerts that belong to the same attack fall into another batch. Their weak classification is false positive, as they are actually not weak across multiple batches. Ideally, it is left up to the correlation algorithm $\mathscr{K}$ to compensate such cases or alternatively tolerate them in the generation of weak meta alerts. The more interesting cases for APT detection are the true positive classifications in lanes 4-5. Here, alerts for one attack are very low in volume and spread over multiple batches, or it is a single alert (or very small group) which might either be a rare outlier or belong to a single shot attack, like a malicious one-time download.

For traditional alert correlation with $\mathscr{K}$, false negatives, i.e., relating not every relevant alert to the attack (lanes 1-3), might be accepted as long as the attack is actually detected. However, accepting false negatives for correlating weak alerts (lanes 4-5) might likely result in the APT attack to go unnoticed. Thus, related weak alerts must be correlated, even if they are temporally dispersed. But in contrast to $\mathscr{K}$ that performs on a series of finite alert batches, weak alerts have to be correlated in a steady stream of new alerts, i.e., across batches. Thus, in the context of stealthy and long-standing APT attacks, weak alerts have to be identified and continuously be assembled to attack steps.

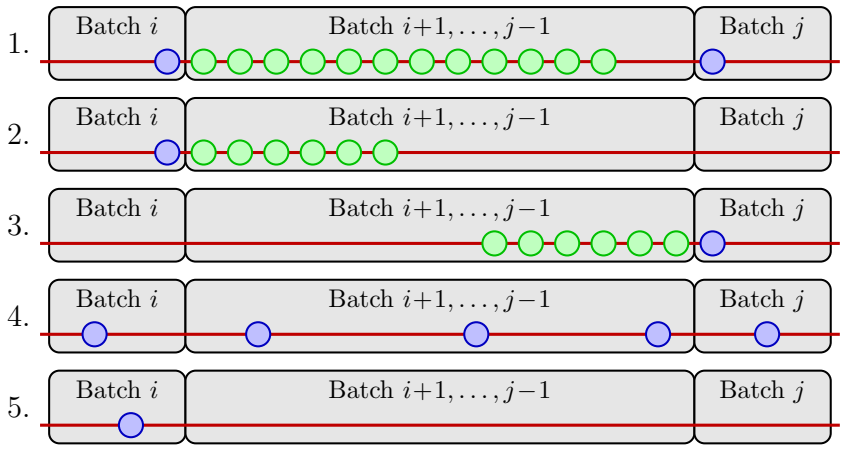


Figure 5.5: Five example cases how the spreading across batches for alerts of an attack results in different classifications regarding weakness.

5.2.2.2 Weak Alert Aggregation

According to the filtering of weak alerts in Section 5.2.2.1, a correlation function $\mathscr{K}$ identifies related alerts in an alert set A that are grouped into clusters. It leaves unclustered alerts as weak alerts W. This correlation function is assumed to run on batches of alerts, which might results in a few weak alerts per batch. However, the challenge is to correlate weak alerts across many batches. For that, the following describes the continuous aggregation of weak alerts.

Characterizing Weak Alerts In the context of APT attacks and the characteristic of lateral movement (cf. Section 2.1.2), alerts that encompass two different network zones are of particular interest. Respective alerts are identified utilizing a *host- & zone-communication graph (HZCG)* as shown in Figure 5.6. The nodes in such a directed graph are source and destination IP addresses of weak alerts, annotated with the respective zone the IP belongs to. The edges indicate the alert direction from one to another zone and summarize the alerts between the two respective IP addresses. In the illustrated example, there are four attackers within `Zone_1` and two targets within `Zone_2`. The targets partly overlap, as the attackers 172.17.0.1 and 172.12.0.2 both attack the two targets, while attackers 172.17.0.3 and 172.17.0.4 each attack a single target only.

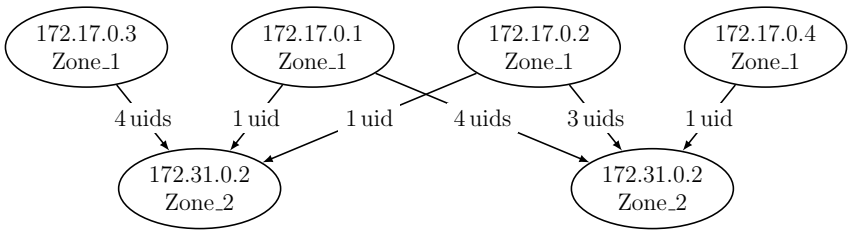


Figure 5.6: Example for weak alerts between two network zones in a HZCG.

Another fundamental characteristic of attacks is whether they are distributed, both in sense of attackers and targets. An attack either encompasses exactly two IP addresses or there are multiple addresses for attackers or targets. However, in the context of APT attacks, noisy attacks with multiple addresses for attackers and targets are likely to cause enough alerts to be reported by traditional approaches. Because of that, the weak alert aggregation focuses on the attack

topologies `OtO`, `OtM`, and `MtO` only (cf. Section 5.2.2.1). The aggregation aims to identify attacks following these patterns among the weak alerts in the HZCG. However, as probably several unrelated attacks are going on from and to different zones as well as false positive weak alerts exist, additional attention has to be paid to the separation between zones and individual attacks.

Aggregating Weak Alerts The aggregation function itself can be thought of as a graph-based *group-by* function on the HZCG. Each node in the graph is checked and its incoming and outgoing edges are iterated separately. Neighbors of each node are grouped by their zone. For example, Figure 5.6 shows that 172.17.0.1 has five alerts to `Zone_2`. The respective neighbor group of node 172.17.0.1 includes the two nodes with addresses 172.31.0.2 and 172.31.0.1 as targets. More specifically, a *neighbor group* consists of objects, each summarizing a neighbor by its IP address and a count of associated alerts:

$$neighbor\_summary := \left\{ \begin{array}{c} \text{IP} : IP\_Address \\ \text{Alerts} : count(alerts) \end{array} \right\}$$

In the example HZCG in Figure 5.6, six *neighbor groups* exist, one for each node. Finally, *neighbor groups* are accumulated per zone to so-called aggregations. An aggregation *agg* carries information about the *time_span* between the first and last alert, the *direction* of the two zones, and topology *label* (`OtO`/`OtM`/`MtO`). Furthermore, an aggregation describes *attackers* and *targets* in the form of neighbor groups but at least one of them consist of a single object only:

$$agg := (time\_span,\ direction,\ label,\ attackers,\ targets)$$

New neighbor groups are calculated whenever a HZCG is constructed, e.g., for each run of the correlation function $\mathscr{K}$ on the next alert batch. Then, they must be merged into existing aggregations. This continuously relates weak alerts across batches to the same attack while it is ongoing. But also neighbor groups within the same batch can be condensed to fewer aggregations.

Inserting Aggregations When a new neighbor group is created, it is directly converted to an aggregation. The collection of aggregations is referred to as database in the following. Any new aggregations have to be inserted into the database of existing aggregations one after another. The pseudocode for the insert function is shown in Algorithm 1. It takes an aggregation as input and returns either the unmodified input, in case no existing and matching aggregation could be found in the database, or returns the merge-result of the input and an existing aggregation object from the database.

Algorithm 1 consists of two outer conditional blocks in Lines 2 and 9. The database is queried depending on the label field of the input aggregation. One-to-one labels must be considered twice. In case of a *one-to-X* label, the database is queried for a single attacker IP (Line 3). Likewise, in case of an *X-to-one* label, the database is queried for a single victim IP (Line 10). A merge is performed if a query result is not empty. The respective field which holds many IP addresses is merged (Lines 4 and 11). The label field is updated accordingly to represent the *many* relation (Lines 6 and 13). The algorithm immediately returns the result whenever a merge is possible. Otherwise, the new aggregation object is inserted into the database without modifications (Line 16).

```
Input: agg: Aggregation, DB: Database with existing Aggregations
Output: Inserted Aggregation
1 begin
2   if agg.label = 'one-to-one' || agg.label = 'one-to-many' then
3     existing_agg ← DB.query(
                                  label = ['one-to-one' OR 'one-to-many']   AND
                          attackers.IPs = agg.attackers.IPs                 AND
                              direction = agg.direction);
      if existing_agg ≠ Null then
4       existing_agg.victims.update(agg.victims);
5       if existing_agg.victims.size ≥ 2 then
6         existing_agg.label ← 'one-to-many';
        end
7       return existing_agg;
      end
8   end
9   if agg.label = 'one-to-one' || agg.label = 'many-to-one' then
10    existing_agg ← DB.query(
                                  label = ['one-to-one' OR 'many-to-one']   AND
                          victims.IPs = agg.victims.IPs                     AND
                            direction = agg.direction);
      if existing_agg ≠ Null then
11      existing_agg.attackers.update(agg.attackers);
12      if existing_agg.attackers.size ≥ 2 then
13        existing_agg.label ← 'many-to-one';
        end
14      return existing_agg;
      end
15  end
16  DB.new_agg(agg);
17 end
18 return agg
```

Algorithm 1: Insert function to form aggregations.

The example in Figure 5.6 results in four aggregation objects after inserting the six neighbor groups according to Algorithm 1. Two `MtO` aggregations exist for the targets 172.31.0.2 and 172.31.0.1, and two `OtM` aggregations exist for the attackers 172.17.0.1 and 172.17.0.2. Event though a target is attacked by multiple attackers or an attacker attacks multiple targets, it does not necessarily mean that all the respective alerts belong to the same attack. Hence, a last step is performed by separating individual attacks among the weak alerts to generate weak meta alerts.

5.2.2.3 Weak Meta Alert Generation

The aggregation procedure groups all those alerts that share the exact same network direction and label. However, the aggregated neighbor groups grow naturally with more weak alerts. The aggregation does not highlight any relations among the weak alerts. Thus, the aggregation process on its own cannot be used for a correlation. The following describes how to leverage the aggregated groups to inspect each aggregation object and to extract smaller clusters of alerts (still in a certain network direction) that are of more interest than others.

The basic idea of clustering weak alerts within an aggregation is to group attackers and targets by their alert frequency. Intuitively, this identifies false positive weak alerts that are only considered weak because of their disadvantageous distribution among alert batches. During aggregation, they eventually stack up and form a large group across batches. However, even true positive weak alerts might falsely be contained in such a stacked-up aggregation group over an extended period of time. They can still be identified and extracted by their lower frequency.

To identify groups of alerts with different frequencies, density-based clustering is leveraged. Such clustering algorithm benefit from how their parameters are used. In contrast to other unsupervised clustering techniques, a density-based algorithm like density-based spatial clustering of applications with noise (DBSCAN) does not require any parameterization regarding cluster sizes or amount of clusters. Instead, DBSCAN is able to extract clusters from an unknown corpus, solely based on the density of the appearance in the search space. The parameters of interest are min_pts and eps. These parameters define how many points are at least required to form one cluster (min_pts) and how big the distance between two points can be at most for them to be grouped to the same cluster (eps). Applied to the weak alert aggregations, the distance between two neighbors is defined by the difference between their alert counts in the respective neighbor summaries. This can be seen as a one-dimensional space in which for each IP address one data point, with the value of the alert count, exists. Intuitively, clusters with the lowest alert frequency might be the most stealthy ongoing attacks. To detect even the extreme stealthy attacks, alerts are not filtered by clustering but should end up in clusters with at least one alert. The pseudocode in Algorithm 2 generates weak meta alerts based on input aggregations.

The generation function starts by iterating all aggregation objects at the beginning of Algorithm 2 in line 2. The algorithm body is divided into three major blocks. DBSCAN has to run with different arguments, depending on the label of the currently iterated aggregation. In case it is a *one-to-many* label (Line 4) the victims form the *many* entity are subject to density-based clustering (Line 6). Similarly, the attackers are subject to density-based clustering in case of *many-to-one* relations (Lines 18 and 20). A new weak meta alert is generated for each cluster that results from DBSCAN. Therefore, the label is re-calculated based on the actually involved IP addresses (Lines 8-11 and 22-25). Similarly, attackers and victims are re-calculated in Lines 12-13 and 26-27. The frequency calculation is defined straightforward. The number of involved IP addresses in each cluster is divided by the total number of IP addresses in the original aggregation object (Lines 14 and 28). Lines 32-37 show the special case that the aggregation was already labeled with *one-to-one*. It is not possible to break down the frequency relations of the involved IP addresses any further.

```
Input: Aggs: List of Aggregations
Output: W_MA: List of Weak Meta Alerts
begin
    for agg in Aggs do
        dir ← agg.direction;
        if agg.label = 'one-to-many' then
            total_alerts ← agg.attackers.Alerts.size();
            clusters ← DBSCAN(agg.victims);
            for cluster in clusters do
                label ← 'one-to-one';
                if cluster.IPs.size() ≥ 2 then
                    label ← 'one-to-many';
                end
                attacker ← agg.attackers[0].IP;
                victims ← cluster.IPs;
                freq ← cluster.Alerts.size()/total_alerts;
                w_ma ← (label, dir, alerts, attacker, victims, freq);
                W_MA.add(w_ma);
            end
        else if agg.label = 'many-to-one' then
            total_alerts ← agg.victims.Alerts.size();
            clusters ← DBSCAN(agg.attackers);
            for cluster in clusters do
                label ← 'one-to-one';
                if cluster.IPs.size() ≥ 2 then
                    label ← 'many-to-one';
                end
                victim ← agg.victims[0].IP;
                attackers ← cluster.IPs;
                freq ← cluster.Alerts.size()/total_alerts;
                w_ma ← (label, dir, alerts, attackers, victim, freq);
                W_MA.add(w_ma);
            end
        else
            label ← 'one-to-one';
            victim ← agg.victims[0].IP;
            attacker ← agg.attackers[0].IP;
            alerts ← agg.attackers[0].Alerts;
            freq ← 1.0;
            w_ma ← (label, dir, alerts, attacker, victim, freq);
            W_MA.add(w_ma);
        end
    end
end
return W_MA
```

Algorithm 2: Generating weak meta alerts.

5.2.3 Summary of Alert Clustering

To reduce the alert volume, two complementing alert clustering algorithms have been presented that isolate alerts from different attacks and summarize respective alerts as meta alerts. These meta alerts represent individual attacks together with all the alerts they caused. The identification of related alerts in clustering leverages that alerts from the same attack are likely to have some equal feature values. This is especially true for the network-wide attacks such as distributed attacks with a lot of similar malicious activity.

For the detection of such bulk attacks, this section has presented a graphed-based algorithm that transforms the alerts into a *alert similarity graph*. Alerts are presented as nodes in this graph, and edges indicate a potential relation between two alerts that is determined by the similarity between the alerts' features. Within this graphs, community clustering is performed to identify well-connected subgraphs. Each subgraph is then supposed to represent the alerts of an individual attack. The accuracy of isolating alerts from different attacks is evaluated in Section 6.5.1.2 and tested on real-world data in Section 6.5.2.2.

In contrast to bulk attacks, where alerts are likely to be clustered by the graph-based algorithm, stealthy APT-like attacks cause infrequent alerts that get lost in the shuffle. While the alerts from bulk attacks form well-interconnected subgraphs that become apparent because of their size, the visibility of relations among the infrequent alerts seems weak, which is why they are denoted as weak alerts. These weak alerts are unlikely to fall in the same alert batch because of the large alert volume from other attacks in between. Consequently, the graph-based algorithm on the basis of alert batches cannot cluster the weak alerts in the first place.

The clustering of weak alerts, therefore, requires a continuous processing of alerts across batches. This section has presented a weak alert correlation algorithm that consumes alerts that remain unclustered by the graph-based alert clustering. The algorithm continuously aggregates these weak alerts until they assemble to a weak meta alert. Thus, the algorithm first transforms the unclustered alerts of every new batch into a *HZCG* that summarizes the weak alerts regarding their IP addresses and network zones. Across alert batches, the algorithm maintains aggregations of weak alerts that are updated with every new graph. Based on the continuous aggregation, the relations among IP addresses from the weak alerts become apparent as more and more related alerts are aggregated during the stealthy and long-lasting APT attack. When this is the case, density clustering is performed to extract the alerts that assemble the pattern of an APT attack step. Section 6.6 investigates to which extent the clustering of weak alerts can supplement the graph-based clustering to identify stealthy attack steps.

5.3 Attack Interconnection with Context

The second and third stage in the alert correlation process (cf. Section 5.1) supplement the clustered network alerts from Section 5.2 and link the alerts from different attacks and their steps, respectively. This linking is required, because an isolated view on the clustered alerts eventually underestimates the network-wide attack. In fact, two clusters might actually should be seen together because of either of the two dimensions: First, an attacker performs multiple attacks against different network areas step-by-step to achieve the overall attack goal. Second, an attacker widely distributes the same attack, e.g., to different network sites with individual IDSes, such that no IDS captures the full attack but only parts of it.

To assemble attack relations despite these challenges, attack interconnection in this section leverages additional context. Context supplementing for the approaches here is based on characterizing the who-targets-whom structure as a representation of the attack scenario. In addition to the concrete feature values among the alerts, interconnection benefits from this value-independent characterization of the attack scenario. This context is derived from the alerts themselves and does not require an external knowledge base.

This section combines revised parts of the two conference papers [HF18] and [HWF19]. The section presents two attack linking approaches for two different purposes. The first approach identifies multi-step attacks, in which the steps build upon each other by exploiting one or multiple targeted hosts to make use of them in the subsequent step. The second approach enables the collaboration of different network sites by identifying that two sites have potentially become a victim of the same attack.

5.3.1 Graph-based Multi-Step Detection

The first approach identifies attack steps that are part of a multi-step attack. The approach first derives additional context about the attack scenario from the alerts of each attack step. The attack scenario is afterwards utilized to assemble multi-step attacks based on equal IP addresses between alerts from different attack steps.

5.3.1.1 Scenario Identification

The supplementing attack scenario is derived from the clusters of alerts that represent individual attack steps. The goal is now to characterize the attack scenario of each cluster. For that, the communication patterns between attackers and victims are determined. Especially distributed attacks such as DDoS, port scans, and worms are of interest here, as they involve many systems at the same time and thus result in a large number of different alerts.

Communication Topology Most alerts in a cluster are usually similar to each other and thus well interconnected in G_{attr} (cf. Section 5.2.1). Hence, such a graph structure is inappropriate to identify attack scenarios. Thus, per identified cluster $C_i \in \hat{C}$, a less dense *alert flow graph* $G_{\text{flow}} = (V, E)$ is established with the nodes V to be the set of all source and destination IP addresses in a cluster C_i. An edge $(u, v) \in E$ in the graph between two nodes $u, v \in V$ exists, if there is an alert $a \in C_i$ with the source node u as attacking IP and the destination node v as target IP. The direction of an edge therefore indicates who attacked whom.

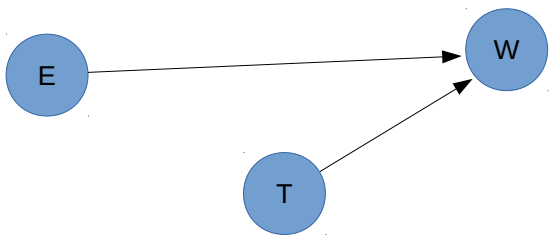


Figure 5.7: An exemplary alert flow graph for three hosts.

Figure 5.7 shows the graph G_{flow} for the alerts in Figure 5.3a. In this example, the three alerts in the graph G_{attr} in Figure 5.3b are assumed to belong to the same cluster. G_{flow} contains the IP addresses E, T and W as nodes and edges from E and T to W, as the node with IP address W is attacked by nodes with the addresses E and T.

To identify attack scenarios, the nodes' degree in $G_{\text{flow}} = (V, E)$ is used in a scheme for distributed attacks involving either multiple sources, destinations, or both. This scheme characterizes nodes $v \in V$ as follows: An *attacker* has an outgoing degree ≥ 1 and a *victim* has an incoming degree ≥ 1. Moreover, in distributed attack scenarios and multi-step attacks, a node can be attacker and victim at the same time. Four attack scenarios are distinguished and further summarized in Table 6.12:

- One-to-One `OtO`: One source is attacking a single destination, which is a special case of the other scenarios.
- One-to-Many `OtM`: One source is attacking multiple destinations, e.g., scanning a subnet. Characteristic for all alerts is the same attacking source IP.
- Many-to-One `MtO`: Multiple sources are attacking a single destination, e.g., in a DDoS attack. When attacking a specific service, all related alerts might have the same destination IP and port.
- Many-to-Many `MtM`: Multiple sources and destinations are involved, e.g., like in a worm spreading. Since such worms spread by targeting a specific application, alerts in this scenario will have at least the same destination ports.

Topology Heuristics To identify the scenario of a given attack, four metrics describe how good a cluster $C_i \in \hat{C}$ matches one of the given attack scenarios. The four metrics $\delta_{\texttt{OtO}}, \delta_{\texttt{OtM}}$, $\delta_{\texttt{MtO}}$, $\delta_{\texttt{MtM}}$ indicate the certainty between $[0,1]$ for a match with the scenarios `OtO`, `OtM`, `MtO`, and `MtM`, respectively. These metrics become 1 if a scenario matches perfectly and they are smaller than 1 if the scenario and C_i do not match exactly. Each metric consists out of three equally weighted summands that describe the three ratios of attacker number $|A|$, of target number $|T|$, and of the difference $|\,|A|-|T|\,|$ all to the expected value in the respective scenario. The formulas are constructed, so that there is a linear relationship between the respective scenario matches completely ($\delta = 1$) or only partially ($0 < \delta < 1$).

A cluster that perfectly matches scenario `OtO` contains two nodes $|V| = 2$, one target $|T| = 1$ and one attacker $|A| = 1$. In that case, all three summands for $\delta_{\texttt{OtO}}$ in Equation 5.2 become one. When more targets or attackers are involved, the metric degrades and converges towards zero. The scenario `OtM` expects one attacker and $|V|-1$ targets, which results in a difference of $|V|-2$. Therefore, the first summand of metric $\delta_{\texttt{OtM}}$ in Equation 5.2 becomes 1 if $|A| = 1$ and the second summand is 1 for $|T| = |V|-1$. This results in a difference between attackers and target of $|V|-2$. $\delta_{\texttt{OtM}}$ becomes closer to 0, if there are more attackers or less targets. Analogous, the scenario `MtO` ($\delta_{\texttt{MtO}}$ in Equation 5.2) expects $|V|-1$ attackers and one target, which results in a difference of $|V|-2$. A perfectly matching `MtM`-cluster ($\delta_{\texttt{MtM}}$ in Equation 5.2) needs to have $|V|$ attackers and $|V|$ targets, so that the difference between them is zero.

$$
\begin{aligned}
\delta_{\texttt{OtO}} &= \frac{1}{3} \cdot \left(\frac{|V|-|A|}{|V|-1} + \frac{|V|-|T|}{|V|-1} + \frac{|V| - |\,|A|-|T|\,|}{|V|} \right) \\
\delta_{\texttt{OtM}} &= \frac{1}{3} \cdot \left(\frac{|V|-|A|}{|V|-1} + \frac{|T|}{|V|-1} + \frac{|\,|A|-|T|\,|}{|V|-2} \right) \\
\delta_{\texttt{MtO}} &= \frac{1}{3} \cdot \left(\frac{|A|}{|V|-1} + \frac{|V|-|T|}{|V|-1} + \frac{|\,|A|-|T|\,|}{|V|-2} \right) \\
\delta_{\texttt{MtM}} &= \frac{1}{3} \cdot \left(\frac{|A|}{|V|} + \frac{|T|}{|V|} + \frac{|V| - |\,|A|-|T|\,|}{|V|} \right)
\end{aligned}
\tag{5.2}
$$

The highest metric determines the scenario and the corresponding label $l \in L = \{$`MtO`, `OtM`, `MtM`, `OtO`$\}$. The certainty of scenario identification and label assignment is:

$$\delta = \max(\delta_{\texttt{OtO}}, \delta_{\texttt{OtM}}, \delta_{\texttt{MtO}}, \delta_{\texttt{MtM}})$$

5.3.1.2 Attack-Step Correlation

For the detection of multi-step attacks, all identified clusters $\hat{C}$ are first transformed into a directed labeled graph that is denoted as *attack similarity graph* $G_{\text{over}} = (\hat{C}, E)$. An edge $(C_i, C_j) \in E$ is included in G_{over} if the two clusters belong to the same multi-step attack.

The linking of attack steps utilizes the labels $l_i, l_j \in L$ of the two clusters $C_i, C_j \in \hat{C}$ to derive a tag `Many` or `One` for the attackers and targets For example, when a cluster is labeled with `OtM`, the set of attacking IPs is tagged with `One` and the set of target IPs is tagged with `Many`.

Then, the approach compares the set of attackers from both clusters (sim_{AA}), the set of targets from both clusters (sim_{TT}), the set of attackers of C_i with the set of targets of C_j (sim_{AT}), and the set of attackers of C_j with the set of targets of C_i (sim_{TA}). For each of the four host comparisons, the host sets X, Y can be attackers or targets. Their tags depend on the tags of the clusters and on the specific comparison. The calculation of the specific host similarity uses one of the following metrics. First, the *Jaccard metric* $J(X,Y) = \frac{X \cap Y}{X \cup Y}$ is used when the two compared sets of hosts X, Y are expected to be equal, i.e., both are tagged equally. Second, the *overlap coefficient* $S(X,Y) = \frac{X \cap Y}{X}$ is used when one set of hosts X is expected to be part of the other set Y, i.e., their tags are different. In this case, X is the set that is tagged with `One` and Y is the one tagged with `Many`.

The comparison of two clusters C_i, C_j with each other computes the four values as mentioned above and then takes the maximum of it $\text{sim} = \max\{\text{sim}_{\text{AA}}, \text{sim}_{\text{TT}}, \text{sim}_{\text{AT}}, \text{sim}_{\text{TA}}\}$. If sim exceeds a predefined threshold σ, an edge (C_i, C_j) with label sim is added to G_{over}. This has to be done for all cluster combinations from set $\hat{C}$. As a result, G_{over} contains the relations among all identified clusters that are above threshold σ. Clusters as part of multi-step attacks then are located on a path connecting them within this graph.

5.3.2 Collaborative Attack Correlation

While the graph-based multi-step detection from Section 5.3.1 requires attacks to be represented by clusters containing all related alerts, the alerts are spatially dispersed in case of a distributed IDS deployment, i.e., a collaborative intrusion detection system (CIDS) [Vas+15b]. Thus, the distributed alert clusters that result from clustering alerts on each IDS locally have to be merged before being processed by the graph-based multi-step detection. However, collection and clustering all alerts centrally does not scale and might be infeasible for large networks. Therefore, the following approach for attack correlation performs a fine-grained characterization of alert clusters regarding their attack scenario and efficiently identifies alert clusters that potentially result from the same attack across different IDSes.

5.3.2.1 Comparing Attack Characteristics

The basic idea of this correlation algorithm is to transform a set of alerts into a much smaller representation that conserves structural characteristics of attacks. For that, network motifs, specifically the so-called motif signatures (cf. the motif-based anomaly detection in Section 3.4.1), are a perfect candidate to summarize the communication structure of the hosts involved in an attack. This allows for a comparison of structural characteristics of attacks, even without prior knowledge of these characteristics and mostly independent from the attack size.

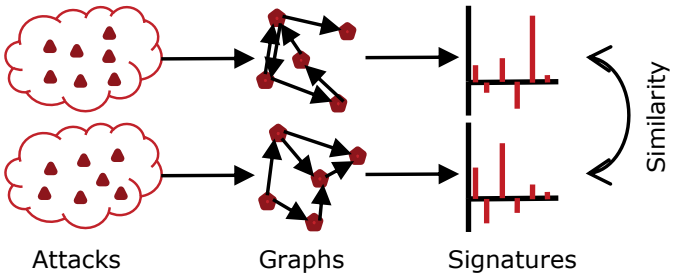


Figure 5.8: Schema to compare the alerts of two attacks by their motif signatures.

Input to this approach for attack interconnection are clustered alerts $C_i \in \hat{C}$ as from the alert clustering with graph-based community clustering and weak alert correlation (cf. Section 5.2) or others (cf. Section 3.4.2). Referred to as attacks, alerts clusters are the input to the collaborative attack correlation, as illustrated by the schematic overview in Figure 5.8. The following paragraphs describe the next steps in detail, which is the transformation of attack data into graphs, the calculation of motifs signatures, and their comparison.

Transformation of Attacks to Graphs The approach assumes that attack characteristics manifest in the structure of the communication graph that contains the malicious communication relations among involved hosts. This graph is derived from all alerts $a_i \in C_j$ of attack $C_j \in \hat{C}$, where an alert a_i has several attributes. From all attributes defined in Section 5.1, the collaborative attack correlation requires only an alert description $a_i = (S{:}T \rightarrow D{:}L)$ with source IP S and source port T and with destination IP D and destination port L. Based on these four attributes of alerts, the approach generates a *communication structure graph* G_{com} for all alerts of an attack C_j. In $G_{com} = (V,E)$, nodes $v \in V$ represent either a host by its IP address or the port on a

specific host. The edges reflect who attacked whom and whether a particular port or several different ports are used in the attack.

To build the graph G_{com} for a specific attack C_j, all alerts $a_i \in C_i$ are added to the graph consecutively. For that, the set of nodes V is extended by nodes representing the hosts, i.e., $\{S,T\}$, and nodes representing their ports, i.e., $\{S{:}T, D{:}L\}$. This notation ensures that ports are always bound to hosts. To reflect who attacked whom, the edges $\{(S,S{:}T),(S{:}T,D{:}L),(D{:}L,D)\}$ are added to E in G_{com}. Intuitively, this describes what is visualized in Figure 5.9, the port and IP used to attack another IP on a respective port.

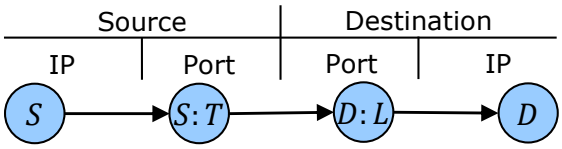


Figure 5.9: Adding an alert ($S{:}T \rightarrow D{:}L$) to graph G_{com}.

Calculation of Scenario Signatures To get from the alerts in C_j via the graph G_{com} to a smaller abstraction, the motif signature of the graph is calculated. For that, all subgraphs $G' = (V',E') \subseteq G_{com}$ of size n, i.e., $|V'| = n$, are numerated, and for each the specific motif pattern m_i with $i \in [0,N-1]$ is determined (cf. the motif-based anomaly detection in Section 3.4.1). Counting the number of occurrences for all motifs m_i results in a *fingerprint* of the graph. The vector that contains the absolute number of occurrences of every m_i is denoted as motif signature F^A.

As F^A is directly dependent on the graph size $|G_{com}|$, it does not allow to compare the structure of two graphs that are of different sizes. To allow such a comparison, the authors in [Mil+02] introduce the so-called Z-Score. It uses the signature F^A of graph G_{com} to calculate for every m_i how much it is over- or underrepresented compared to a random graph of the same size and with the same number of edges as G_{com}. The Z-Score of a specific motif m_i in a graph G is calculated by

$$Z(m_i) = \frac{F^A(i) - F^{rand}(i)}{sd}$$

with $F^A(i)$ being the absolute number of motif m_i in G_{com}, $F^{rand}(i)$ being the average absolute count of motif m_i in respective random graphs, and its standard deviation sd. This is done for every absolute number of motifs in F^A. The resulting motif signature with the Z-Score values is denoted as F^Z. Any motif signature can simply be represented as an array of fixed length, e.g., 16 to cover all possible 3-motifs ($n = 3$).

Comparison of Scenario Signatures Comparing two motif signatures F_1^Z and F_2^Z is supposed to determine how similar they are. Their similarity should be 1, i.e., 100%, if the signatures are equal. However, finding a metric to calculate the similarity in a meaningful manner is not intuitive. The reason is that the values even in the Z-Score signature F^Z are not limited to a fixed range. As the graph size, i.e. attack size, still has an impact on the Z-Score values, the values of every motif in two signatures F_1^Z and F_2^Z cannot directly be compared. Doing so would not achieve high similarity for attacks with similar characteristics but of different size. Therefore, another comparison between two motif signatures is designed in such a way that

similar attack characteristics are identified even if the attacks are of different sizes. For that, a motif that is statistically over- or underrepresented in F_1^Z should also be statistically over- or underrepresented in F_2^Z. Furthermore, the comparison should consider how much a specific motif is over- or underrepresented compared to the other motifs in the signature. The idea is to not have a pairwise comparison of the motif values in F_1^Z and F_2^Z. Instead, the similarity between F_1^Z and F_2^Z reflects how similar the relations among the motif values in F_1^Z are to the relations among the motif values in F_2^Z.

For such a comparison, the Z-Score signatures F_1^Z and F_2^Z are interpreted as vectors $\vec{u}, \vec{v}$, always of fixed length. This make a signature to look like a vector in a multi-dimensional space with the number of dimensions equal to the length of the vectors. The higher the values in a signature, the larger is the vector in the multi-dimensional space. The calculation becomes independent from the vector length when determining the angle between two vectors in the multi-dimensional space. For that, the inner product $< \vec{u}, \vec{v} >$ and the Euclidean norms $||\vec{u}||_2$ and $||\vec{v}||_2$ are calculated for the angle ϕ (Equation 5.3). This finally leads to the similarity $0 \leq sim \leq 1$ (Equation 5.4).

$$cos(\phi) = \frac{< \vec{u}, \vec{v} >}{||\vec{u}||_2 \cdot ||\vec{v}||_2} \tag{5.3}$$

$$sim = \frac{cos^{-1}(\phi)}{\pi} \tag{5.4}$$

Once the alerts of every attack $C_i \in \hat{C}$ are transformed into motif signatures $F_i^Z \in \hat{F}$, the motif signatures of the different attacks, i.e., alert sets, are compared. The goal is to find attacks with the same characteristics. Such a comparison can be calculated efficiently as a motif signature is of a fixed and far smaller size than the respective alert sets. For the attack correlation algorithm, predefined characteristics of attacks can be used to identify specific attack scenarios and to label them accordingly. Alternatively, the attack correlation operates without a knowledge database and learns attack scenarios on its own. It labels attacks according to dynamically derived characteristics of alert sets. Both approaches are described in the remainder of this section. Either way, comparing the Z-Score signatures F^Z to classify attack scenarios requires a threshold τ to require a minimum similarity. Above this threshold, two signatures are assumed to belong to the same attack scenario.

5.3.2.2 Signature-based Classification

The characteristics of already known attack scenarios are defined by reference scenarios $R_x \in \hat{R}$. A reference scenario R_x reflects the characteristics of a specific attack scenario and therefore is representative for all attacks of this attack scenario and their alert sets, respectively. In fact, R_x is just a motif signature F^Z of a typical attack in the attack scenario that should be identified. It is modeled by transforming a representative alert set A into a *communication structure graph* G_{com} and by computing its motif signature F^Z. Hence, the set $\hat{R}$ consists of small motif signatures. The size of this set equals the number of predefined attack scenarios that should be identified.

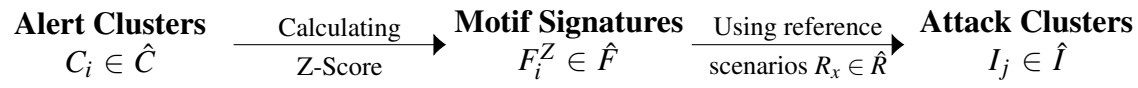


Figure 5.10: Notation for signature-based classification using reference scenarios.

When classifying an attack $C_i \in \hat{C}$, its motif signature F_i^Z is compared to all reference signatures $R_x \in \hat{R}$ using the similarity function described in Section 5.3.2.1. In general, the highest similarity determines the attack scenario that is assigned to attack C_j. However, the minimum similarity threshold τ needs to be respected, because there could be attacks from unknown scenarios that are not included in $\hat{R}$. Hence, they should not be labeled with a known attack scenario. Assigning all attacks $C_i \in \hat{C}$ to reference scenarios results in attack clusters $I_j \in \hat{I}$, as illustrated in Figure 5.10 (cf. Section 5.1). The requirement for attacks of the same scenario $F_i^Z \in I_j$ to have a minimum similarity τ to the respective reference scenario R_x is formalized in Equation 5.5. The requirement of closest match among all reference scenarios $\hat{R}$ is formalized in Equation 5.6.

$$\forall F_i^Z \in I_j : sim(F_i^Z, R_x) \geq \tau \tag{5.5}$$

$$\forall F_i^Z \in I_j \, \forall R_y \in \hat{R} : sim(F_i^Z, R_x) \geq sim(F_i^Z, R_y) \tag{5.6}$$

5.3.2.3 Unsupervised Clustering

Reference signatures are not always available. However, the motif-based comparison of attack characteristics can be used to cluster similar attacks and to dynamically derive reference scenarios $\hat{R}$ for them. This is done by learning new attack scenarios via a hierarchical clustering in two steps. Note that the following two paragraphs for the description of these steps focus on how the motif-based signatures can be used by machine learning to identifying similar alert cluster across different IDSes. In combination with a communication schema, the fundamentals presented here are directly applicable to be used in a CIDS, or they can even be incorporated into network anomaly detection, e.g., to extend the work of Juszczyszyn et al. [JK11].

Hierarchical Clustering The first step to learn attack scenarios clusters all attacks from the same attack scenario. For that, attacks are clustered based on the similarity of their motif signatures F^Z. Generally, the comparison of motif signatures F^Z aims for high similarities among signatures for attacks of the same scenario and low similarities of signatures for attacks from different scenarios. The intention is that clustering the motif signatures of attacks results in attack clusters, one for each detected attack scenario. Hierarchical clustering is most applicable here to cluster attacks into attack scenarios for two reasons. First, it can use the similarity threshold τ (cf. Section 5.3.2.1) as clustering parameter to intuitively control the clustering and its outcome. And second, the visualization as dendrogram allows a manual inspection of the potential clusters depending on τ.

As said, the clustering parameter τ in hierarchical clustering controls the minimum similarity, i.e., maximum distance, for two attacks, so that they are still part of the same attack scenario. Thus, τ must be chosen in a way such that (1) it is low enough to allow attacks of the same scenario to result in one cluster and (2) it is high enough that two attacks of different scenarios do not result in the same cluster. An example for hierarchical clustering of attacks is visualized as dendrogram in Figure 5.11. The x-axis of the figure represents the individual attacks and the y-axis represents $1 - \tau$ as maximum distance for two attacks or scenarios to be merged into the same scenario. Hence, at distance 0, only equal attacks will be merged and at distance 1, all attacks are merged into a single scenario. When stepping from 0 to 1, similar attacks or scenarios are merged once the value on the y-axis reaches their respective distance. The idea in hierarchical clustering is to define a cut-off distance, which determines the final clusters. The

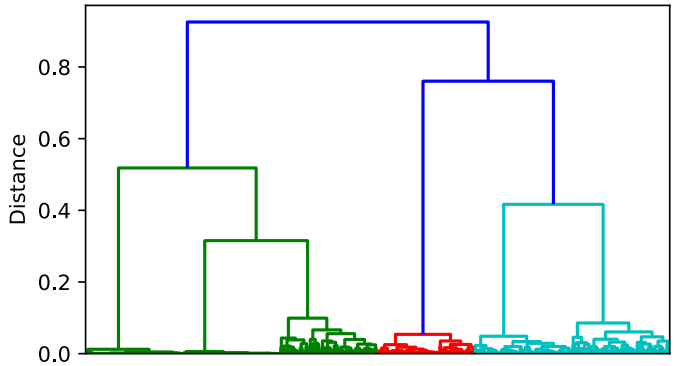


Figure 5.11: Hierarchical clustering for attacks from six example scenarios.

outcome are the clusters, i.e. scenarios, that all the attacks have been merged into at the specific cut-off distance in the dendrogram.

The cut-off distance is $1-\tau$ to form clusters $I_j \in \hat{I}$ of attacks $F_i^Z \in I_J$ with the desired maximum heterogeneity within a cluster. The hierarchical clustering of F^Z for the set of attacks merges clusters with respect to the maximum distance within the resulting cluster, which is known as the `complete` method [Sor48]. This means that attack scenarios are defined by the maximum distance between contained attacks, which can be formally stated as in Equation 5.7.

$$\forall I_j \in \hat{I}: \quad \forall F_1^Z, F_2^Z \in I_j \; sim(F_1^Z, F_2^Z) \geq \tau \tag{5.7}$$

Deriving Reference Scenarios The attacks in a cluster $F_i^Z \in I_j$ formed by hierarchical clustering are supposed to belong to the same attack scenario because of their similar characteristics. To actually extract the characteristics for each attack cluster, a motif signature F^Z is derived per cluster as reference scenario $R_x \in \hat{R}$ to represent the new attack scenario $I_j \in \hat{I}$. Instead of constructing a signature for the attack cluster synthetically, an existing signature from the cluster should be chosen that best represents all other signatures in the cluster is chosen. More specifically, it should be the one with the highest similarity to every other attack on average. This is formalized in Equation 5.8.

$$\forall F_i^Z \in I_j: \quad \sum_{y=0}^{|I_j|} sim(F_i^Z, F_y^Z) \leq \sum_{y=0}^{|I_j|} sim(R_x, F_y^Z) \tag{5.8}$$

After these two steps, a set of attacks, i.e., a set of alert sets, is represented by a number of reference scenarios that is controlled by the similarity threshold τ.

5.3.3 Summary of Attack Interconnection

To interconnect attacks, two attack correlation algorithms have been presented that identify similarities among the attacks based on their alerts. For that, the attack correlation algorithms consume the meta alerts, i.e., the output from alert clustering (cf. Section 5.2). The identification of related attacks for attack interconnection leverages additional context about the attacks. The

two presented algorithms determine the attack scenario and the who-attacked-whom characteristic in particular. They, however, define this characteristic in different ways and use it for two different purposes.

The attack scenario in the first algorithm is defined by a simple model that distinguishes four classes of the who-attacked-whom structure. On the level of hosts, this model labels the number of attackers as well as the number of targets with either *One* or *Many*. Labeling an attack with this scheme results in one of four possible combinations, i.e., the attack scenario. The scenario label is incorporated into the linking of attacks when two of them have the same host involved. The robustness of this label against false positive alerts is evaluated in Section 6.5.1.3. Especially in distributed attacks, quite a number of hosts might be involved in the attack. However, the attacker might follow up on only a small number of these hosts. Comparing not only the hosts' IP addresses among attacks but also the scenario label, assists security operators to identify the important relations among attacks when assembling them to multi-step attacks. The result of this multi-step attack correlation is a graph that concisely summarizes individual attacks as nodes and highlights relations among them as edges labeled with the attack scenario. Section 6.5.2 investigates to which extent the results can help to detect multi-step attacks in the real world.

Apart from detecting multi-step attacks, the second attack correlation algorithm in this section interconnects meta alerts that potentially describe the same attack. This is necessary for large networks that run more than one IDS, or for Internet-wide attacks that hit more than one network site. Either way, each IDS captures the attack only partly, and a common view on this attack among all IDSes is required to be established. Instead of exchanging and comparing all alerts, the attack correlation algorithm identifies candidates for the same attack on the basis of meta alerts. This enables to efficiently exchange small attack summaries and to afterwards compare alerts of candidate attacks only.

The attack summary in this second attack correlation algorithm differs from most algorithms that search for common attributes among the alerts from different attacks. These approaches can reveal if two victims are targeted by the same attacker but they cannot tell if the attacker performs the same kind of attack in both cases. Because of this, the presented attack summary is based on network motifs [Mil+02] to fingerprint attacks. This fingerprint abstracts the characteristic of the attack scenario, considering not only the relations among the involved hosts but also the ports they use. The accuracy of differentiating between attack scenarios based this abstraction is evaluated in Section 6.7.1. The resulting abstraction can be magnitudes smaller than the corresponding alert data and allows for a faster comparison of attacks. With the help of this motif abstraction, the algorithm identifies known attack scenarios and is even able to learn previously unknown scenarios. It can be deployed in a centralized or decentralized manner. To which extent the motif-based abstractions can lower the overhead for exchanging alert information as well as its performance on real-world alerts, is evaluated in Section 6.7.2.

5.4 Summary of Network Alert Correlation

Attack detection by correlating IDS alerts is challenged by the large alert volume, which results in missing the big picture of the attacks and choosing suboptimal mitigation actions. The approaches presented in this chapter assist in assembling alert information for an accurate attack summary. Their deployment is not exclusive but should be combined along the alert correlation process as part of the intrusion detection process (cf. Section 3.2.1) to be most

effective. This way, several measures can be implemented that work in two different dimensions. These dimensions are derived from the insight that some alerts describe the same malicious root cause, while others describe different kind of malicious activity and stem from consecutive attack steps.

Thus, the first stage in the alert correlation process achieves a volume reduction by identifying alerts that describe the same attack and clustering them to meta alerts. Apart from one-shot attacks with a single alert that results in a single meta alert each, the reduction in alert volume becomes apparent especially for network-wide attacks such as distributed attacks. Instead of having an IDS alert for every malicious network flow or every involved host of the distributed attack, these related alerts are abstracted by a single meta alert, i.e., a cluster of alerts that represents a single attack. After the transition to meta alerts, the alert correlation process supplements them with addition context before linking the supplemented attack representations for the identification of attack interconnections.

An end-to-end implementation of the full alert correlation process has been presented by *graph-based alert correlation* (GAC) that consists out of a graph-based community clustering for the detection of distributed attacks and the graph-based attack interconnection for the detection of multi-step attacks. GAC achieves alert clustering by identifying alerts that share similar features among each other. GAC then assigns one of four labels to each identified attack to determine the class of distributed scenario. These labeled attacks are used by GAC afterwards when linking attacks steps that involve the same hosts. The resulting multi-step attack graph summarizes individual attacks and highlights relations among the attack steps with additional context about the attack scenario. The scenario context enables to precisely describe the relations among the involved hosts across different network- and non-network-wide attack steps. GAC and its steps are extensively evaluated in Section 6.5.

Apart from clustering of alerts via GAC, this chapter has presented an alternative correlation algorithm for slow and stealthy APT attacks. These attacks cause temporally dispersed alerts that are denoted as weak alerts because a relation among these alerts is weakly visible, if visible at all. More likely, weak alerts get lost in the shuffle of daily alerts and are filtered by GAC as they look like unrelated or false positive alerts. As regular alert clustering overlooks these weak alerts because of their infrequent occurrence, the algorithm for *weak alert correlation* continuously aggregates them over a long time until they assemble to an APT attack step. For that, the correlation algorithm leverage some APT characteristics that become visible in the alert data, including the expansion to another network zone during lateral movement and a steadily stealthy behavior even when involving multiple hosts. Once the aggregation of weak alerts evolves to an attack step, the respective weak meta alert can be fed back to GAC for the purpose of detecting multi-step attacks. To which extent the weak alert correlation can complement GAC, is evaluation in Section 6.6.

Apart from linking of attacks via GAC, this chapter has presented another correlation algorithm for the identification of similar attacks across network monitors. An attack that is not captured by a single IDS but by multiple IDSes in parts, causes spatially dispersed alerts. Without establishing a global picture of the attack, its consequences are tried to be mitigated independently for every monitor instead of collaboratively working on a mitigation for the whole network. To efficiently identify similar attacks without exchanging each and every alert, the motif-based *collaborative attack correlation* exchanges and compares small fingerprints of the meta alerts. Instead of summarizing the alert features, this fingerprint abstracts the attack scenarios regarding the who-attacked-whom structure. This allows to identify similar attacks in a privacy-friendly

manner, even when different sources are used per targeted network site. For a fine-grained characterization of the attack scenario, however, not only the relations among the hosts but also the usage pattern of ports is incorporated into the fingerprint. The accuracy of this collaborative attack correlation is evaluated in Section 6.7.

In combination, the different alert correlation algorithms transform raw IDS alerts into a concise attack representation. In particular, the correlation overcomes the challenges of temporally and spatially dispersed alerts. As a result, the correlation algorithms assemble alerts from distributed and stealthy APT attacks. Furthermore, the algorithms link alerts from related actions to reconstruct multi-step attacks.

6 Evaluation

This chapter conducts a common evaluation of all detection approaches presented in this thesis. Although each approach was presented to solve a particular challenge (cf. research questions Q1-Q5 in Section 1.1), only the combination of them covers the requirements (cf. Section 3.1) for a complete intrusion detection process (cf. Section 3.2.1).

Section 6.1 first highlights the dependencies among the detection approaches presented in this thesis. This first section also lists the tools and data sets that are utilized throughout this evaluation chapter to build a common evaluation pipeline. Then, Section 6.2 qualitatively discusses the ensemble of proposed approaches. Afterwards, the individual contributions to this overall detection pipeline are evaluated. Section 6.3 evaluates the correlation platform zeek-osquery for the joint monitoring of hosts and their network communication. Section 6.4 evaluates how the detection of the network- or Internet-wide threats scan campaigns and botnets can be based on correlated network data in particular. Afterwards, Section 6.5 evaluates the assembling of IDS network alerts for the detection of attackers who target the network either broadly diversified through a distributed attack or in depth through lateral movement. Sections 6.6 and 6.7 evaluate the correlation extensions for temporally and spatially dispersed alerts.

6.1 Tools and Data Sets in the Evaluation Pipeline

The overall goal of the intrusion detection process from Section 3.2.1 is to analyze events for indicators of intrusion and to transform the intrusion information into an intelligible intrusion summary. On this way, monitored activity is processed and transformed into four different abstraction levels (events, alerts, attacks, intrusion summary), starting with high-volume events and ending in a single concise summary. This is illustrated in Figure 6.1 by using *intrusion detection* for the transformation of events to alerts, *alert correlation* for the transformation of alerts into attacks, and *attack correlation* to summarize all attacks and their relations.

The other five filled boxes around the data abstraction levels in Figure 6.1 represent the five contributions of this thesis (cf. their summaries in Section 1.2):

- *zeek-osquery* for detecting particular scenarios through host-network correlation (cf. Section 4.1)
- *Correlated network communication* for detecting scan campaigns (cf. Section 4.2) and peer-to-peer (P2P) botnets (cf. Section 4.3)
- *Graph-based alert correlation (GAC)* for clustering alerts (cf. Section 5.2.1) and detecting multi-step attacks (cf. Section 5.3.1)
- *Weak alert correlation* for detecting stealthy attack steps of an advanced persistent threat (APT) attack (cf. Section 5.2.2)
- *Collaborative attack correlation* for efficient identification of attacks with similar attack scenarios reported by distributed sensors (cf. Section 5.3.2)

Each of these contributions answers a particular research question Q1-Q5 from Section 1.1.

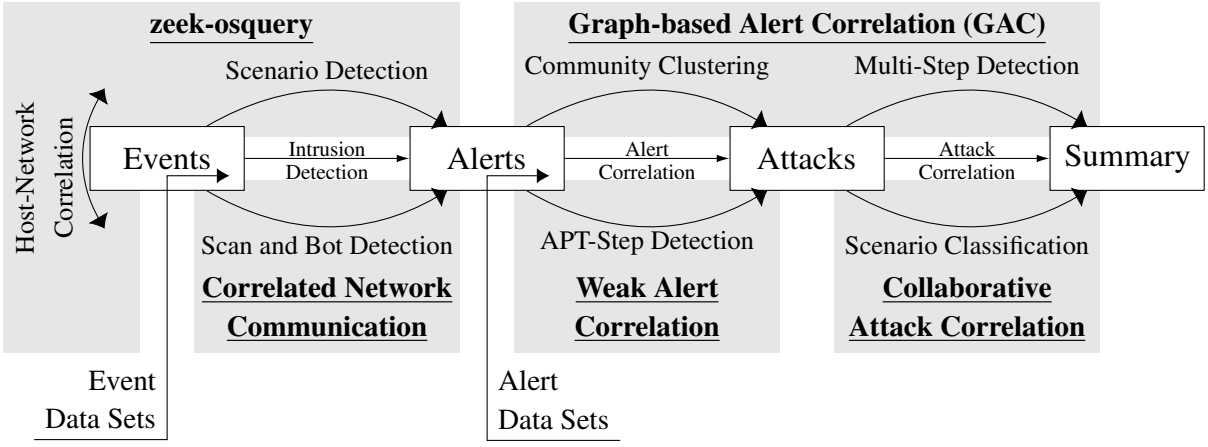


Figure 6.1: Contributions of this thesis along the intrusion detection process

While Figure 6.1 illustrates how the contributions in this thesis are supposed to be combined in an intrusion detection pipeline in practice, the evaluation of each approach is performed individually. For that, different tools and data sets are used throughout this evaluation chapter. They are briefly listed in the remaining paragraphs of this section and described more detailed in the respective evaluation Sections 6.3 to 6.7.

6.1.1 Intrusion Detection Tools in the Evaluation Pipeline

The following paragraphs combine tools and approaches of this thesis along the intrusion detection process for a common evaluation pipeline utilized in this evaluation chapter.

Intrusion Detection The thesis uses the network intrusion detection system (NIDS) *Zeek* (cf. Section 2.4.1) throughout the evaluation to analyze network traffic. It processes the network communication, i.e., the data packets, either live or from a `pcap`-file. The evaluation pipeline can benefit from the Zeek log files in two ways: (1) the detection result of failed network flows, i.e., potential port scans, can directly serve as input for the scan campaign detection. (2) Zeek not only performs intrusion detection but also analyzes the communication for monitoring purposes. Especially the log file about network flows and their statistics serve as input for the P2P botnet detection. Apart from the Zeek log files, the evaluation chapter makes use of the live detection capabilities implemented in Zeek's scripting language. This thesis extends Zeek by some custom detection scripts for the detection of stealthy malware used in APT attacks.

Furthermore, this thesis extends Zeek by a new correlation framework to incorporate host data. For the retrieval of host data, the host monitor *osquery* (cf. Section 2.4.2) is used. It provides an interface to query the hosts for status updates on the level of the operating system (OS), e.g., processes, users, and files. The combination of both tools results in the *zeek-osquery* prototype for the detection of particular attack scenarios by supplementing network flows with additional host context. Both the new correlation framework and the scenario detection of zeek-osquery performs live and is implemented as Zeek scripts.

Alert Correlation The IDS alerts are now required to be correlated to attacks. For that, *GAC* clusters similar alerts from distributed attacks. The resulting alerts clusters, i.e., attacks, are represented by meta alerts. Attacks, therefore, might consist out of alert clusters or just single alerts. Apart from using alert clustering to detect noisy attacks with a high number of alerts in short time periods, *weak alert correlation* works on top by assembling temporally dispersed alerts to APT-like attack steps.

While GAC periodically reports new meta alerts, the continuous processing of the weak alert correlation issues weak meta alerts as soon as the picture of an APT attack step becomes clear. Both types of alert clusters together represent all attacks or attack steps, respectively.

Attack Correlation *GAC* then interconnects the individual attack steps to detect of multi-step attacks. If an attack step, however, comprises not only the local network but also other networks on the Internet, the full picture of the attack at Internet-scale must be established first. This is where the *collaborative attack correlation* comes in, because it efficiently identifies similar alert clusters, i.e., attack, from different network sites that can be potentially merged.

The input to the evaluation of the attack correlation approaches is also alert data. However, the input differs to the large alert sets containing several attacks for the evaluation of alert correlation approaches. For attack correlation, instead, the input alerts represent a particular attack, potentially containing some false positive alerts. Anyhow, data sets with clustered alerts are not available at large scale but only a few possibilities for such sets exist. To still enable a large evaluation of the respective approaches, alternatively large sets of mixed alerts are first clustered by GAC, before they serve as input to the multi-step detection or scenario classification, respectively.

6.1.2 Data Sets for the Evaluation Pipeline

For the isolated evaluation of the individual approaches, the experiments use appropriate data sets. Basically, two different types of data sets are distinguished (cf. Figure 6.1). The event data sets describe low-level monitoring data, while the alert data sets contains the alerts for the evaluation of alert and attack correlation algorithms. The following paragraphs describe the data sets or their sources, respectively.

Event Data Sets The focus of the event data sets is on network communication, partly involving host activities. If possible, the experiments use real-world network captures. Depending on the experiment, an Internet or regular campus site trace is more appropriate. Sometimes, these sets contain the packet payloads, or have only packet headers included. Anyhow, real-world captures are not always available or they simply have no ground truth. In this case, the monitoring takes place in a testbed that is still a real capturing but the activity follows a particular generator to simulate user activity. The following seven data sets are used during the evaluation chapter:

E1: **Real-World Internet Traffic** – Capture of data packets from a backbone link on the Internet between USA and Japan, retrieved from MawiLab [Fon+10]

E2: **Real-World Campus Traffic** – External NetFlow captures of real-world communication on university campus sites, retrieved from University of Twente [Bar+10] and Czech Technical University [Spe+09]

E3: **Real-World Campus Network Monitor** – Live monitoring of a real-world network on campus site, including data packets and host activity

E4: **Real-World Botnet Communication** – The communication relations among bots in a global P2P botnet, retrieved by Strobo Crawler [Haa+16]

E5: **Real-World Attack Traffic** – Various external sample captures of data packets stemming from real-world malware

E6: **Testbed Network Monitor** – Live monitoring in a small network testbed, including data packets and host activity

E7: **Testbed Network Capture** – External monitoring in a large network testbed, including data packets and host logs, retrieved from the University of New Brunswick [SLG18]

Table 6.1 summarizes the characteristics of these data sets for network events. If the data stems from real-world monitoring, the set is labeled with *real-world*. Otherwise, the data is captured still from real systems in a testbed, but the events stem from generated behavior patterns that just mimic real-world situations. Some data sets are labeled *malicious*, if they contain almost only attack data. If also benign data is included but the attack data is explicitly marked, the set is labeled as `mixed`. Furthermore, sets are labeled with *Internet* if they are representative for this scope. Some sets also contain additional monitoring data about the *hosts*. Last, the table also indicates if the data set is *new*, i.e., an outcome of this thesis, and generally refers to the evaluation *sections* that utilize the respective data set.

Set	Real-World	Malicious	Internet	Hosts	New	Sections
E1	yes	no	yes	no	no	6.4.1
E2	yes	no	no	no	no	6.4.2
E3	yes	no	no	yes	yes	6.3.1
E4	yes	yes	yes	no	no	6.4.2
E5	yes	yes	-	no	no	6.6
E6	no	mixed	no	yes	yes	6.3.2 + 6.3.3
E7	no	mixed	no	yes	no	6.6

Table 6.1: Event data sets and their characteristics.

Alert Data Sets Although the approaches for alert and attack correlation are not limited to network data, their evaluation is demonstrated for network alerts only in this chapter. In particular, the following three alert data sets are used during the evaluation chapter:

A1: **Real-World Alerts** – Collaborative collection of alerts for filtered network flows from network sites around the globe, retrieved from DShield[1]

A2: **Real-World APT Attack** – Alerts resulting from analyzing mixed traffic captures based on event data sets E5 and E7

A3: **Artificial Alert Data** – Generated alerts with feature patterns according to particular attack scenarios

1. https://www.dshield.org/

Table 6.2 summarizes the characteristics of these data sets for network events. If the data stems from real-world intrusion detection, the set is labeled with *real-world*. Otherwise, the alerts are artificially generated, i.e., not the detection result of an IDS. Furthermore, sets are labeled with *Internet* if they are representative for this scope. If known, the table gives the underlying traffic source, i.e., on which basis the alert set is created. Last, the table also indicates if the data set is *new*, i.e., an outcome of this thesis, and generally refers to the evaluation *sections* that utilize the respective data set.

Set	Real-World	Internet	Traffic Source	New	Sections
A1	yes	yes	-	no	6.5.2 + 6.7.2
A2	yes	no	E5 + E7	yes	6.6
A3	no	no	no	yes	6.5.1 + 6.7.1

Table 6.2: Alert data sets and their characteristics.

6.2 Qualitative Discussion of Requirements

This section contains a qualitative discussion of the overall detection system containing the contributions of this thesis. The discussion is given according to the requirements from Section 3.1.

Detection Accuracy

The individual detection mechanisms have been designed to solve a particular problem that currently impedes the detection of attacks with network-wide consequences in state-of-the-art solutions. Furthermore, these mechanisms have been developed in the context of being part of an end-to-end intrusion detection process (cf. Section 3.2.1) – from monitoring events to summarizing the attacks. While this might restrict the individual mechanisms slightly in the task they fulfill, it ensures their compatibility when combined in an overall detection system. As a result, the input and output of the mechanisms in sequence are coordinated such that synergy effects arise. This way, a variety of monitoring information is consulted in early stages of the intrusion detection process to strengthen the classification and to verify an intrusion, so that both the numbers of false negatives and false positives decrease. At the same time, the final output of the detection is a concise and short summary of the attacks.

Real-Time Detection

An attack should be detected and reported as soon as possible. In the best case, the security operations center (SOC) is notified about the security breach the same moment the attack is happening. With respect to the intrusion detection process, this would require the whole process implementation to be event-based and every new monitoring event to potentially update the detection summary immediately. Hence, the real-time requirement of individual mechanisms is discussed in the context of the intrusion detection process.

The underlying platform, zeek-osquery, with its correlation pipeline (cf. Section 4.1.3) is indeed event-based. Every retrieved host or network event updates the states and can trigger

a correlation. Similarly, retrieved or self generated alerts can trigger a correlation. Several correlations can exist concurrently, with each meant to implement a detector to produce alerts or to implement an correlation algorithm for alerts or alert clusters to summarize the attacks.

While this design enables a real-time detection in theory, most proposed detectors and alert correlation algorithms are not designed to be event-based. Only zeek-osquery with its continuous stream of host events and its real-time correlation of host and network data, including the demonstrated scenarios (cf. Section 4.1.4), as well as the continuous aggregation of weak alerts (cf. Section 5.2.2.2) are working fully event-based. The other mechanisms are triggered periodically as they require a buffer to be filled with recent events or alerts to run on. Thus, the actual detection might be delayed until the next run of the detector or alert correlation is triggered. Consequently, there is a tradeoff on tweaking the buffer size between minimizing it to reduce the delay and maximizing it to increase detection accuracy.

Nonetheless, the high volume of events and alerts is likely to frequently trigger the detectors and alert correlations. Generally, correlations early in the intrusion detection process are working in real-time or are expected to work at least close to it. Correlations towards the end of the process accumulate the delays of their dependencies, i.e., other correlations and their buffers, and, thus, work in real-time less likely.

Efficiency

To be applicable in practice, the detection mechanisms and their implementations must be resource efficient, e.g., in terms of computational power, memory, and disk space, so that they can run on appropriate hardware. Especially the amount of data being processed together at a particular time directly affects the memory usage and indirectly the computational power. When data processing happens in an event-based fashion, it has the potential to hold the single event only and maybe a few more related information in memory. Then, the input volume for running a detector or correlation is low and the processing is expected to not strain memory capacity too much. Furthermore, processing small pieces of information is also expected to perform fast and, therefore, also not straining the CPU too much.

Along the processing pipeline, many mechanisms follow this design choice. Starting with zeek-osquery, the reconstruction of host events to state and especially the removal of outdated events from the state, keep the memory overhead low. Only up-to-date information remains in memory and any other event or alert information is written to log files. Security tools beyond the scope of the intrusion detection process can make use of those files at a later time. With respect to alert correlation, the definition of meta alerts for the abstraction of several other alerts (cf. Section 5.1) already reduces the complexity of input data and its processing, resulting in a relief of resources. Also the weak alert correlation reduces the data volume when aggregating alerts over time instead of carrying over respective alerts and their full information.

The intrusion detection approaches based on correlated network data (cf. Sections 4.2 and 4.3) have mixed properties with respect to being efficient. First of all, these detectors perform on a large set of network flows. However, the random walk approach for the P2P botnet detection is efficient and the similarities among the flows for the detection of scan campaigns is also easy to calculate. In addition, both detectors are designed to be robust and, therefore, should be able to perform equally well regarding their accuracy when reducing the input number of network flows, e.g., by sampling the NetFlow data.

A very computationally expensive and memory consuming task for alert correlation is the alert clustering with GAC (cf. Section 5.2.1) and the motif calculation (cf. Section 5.3.2.1) for collaborative attack correlation. However, the exchange of motif-based attack signatures at least reduces the message and bandwidth overhead, as the amount of alert data exchanged in a collaborative intrusion detection system (CIDS) is reduced to a few alert summarizes.

Scalability

In zeek-osquery as fundamental security monitoring platform, the system architecture and distributed deployment (cf. Section 4.1.2) is designed to be scalable in many regards. From a network and messaging perspective, the platform scales well with the number of osquery hosts because they can be balanced among multiple proxies. Also the dissemination of messages in the publish-subscribe overlay scales because proxies can aggregate duplicate queries, and because of the overlay itself, a single event can be published to several hosts at once.

Furthermore, different correlations in the correlation pipeline (cf. Section 4.1.3) can be mapped on several authoritative Zeek instances in the overlay. This way, more hardware resources can be added to the overlay to run certain resource intensive detectors or alert correlation algorithms on dedicated Zeek nodes. On the level of particular detectors and correlations, especially the GAC clustering can run in parallel with respect to consecutive alert batches to cope with the large amount of alerts.

By design, a distributed CIDS is supposed to be scalable, and with the collaborative attack correlation (cf. Section 5.3.2), most of the intrusion detection process – up to the attack correlation – can be distributed in the network for having each IDS sensor monitoring a small part of the network. This does not only enable the monitoring of internal network traffic, but also enables the whole detection system to scale with large network sizes.

Easy Deployment

The proposed detection system can be integrated easily into existing environments, because it is based on a NIDS that is usually already deployed by most companies. On top of the existing network monitoring and intrusion detection – independently from signature- or anomaly-based detection techniques – the mechanisms proposed by this thesis can be added to extend the monitoring visibility and to correlate alerts to an attack summary. Also NetFlow monitoring required for the detection of scan campaigns and P2P botnets can be assumed to already exist in company networks. Only the host monitoring via osquery as part of a zeek-osquery deployment potentially requires infrastructure changes, as the network administrator needs to roll out osquery to the hosts in the network. Apart from that, osquery retrieves its configuration, groups, and queries from the centrally managed Zeek backend.

Neither the initial setup, nor the regular operation of the overall detection system requires a particular knowledge base. The maintenance is easy, because the detection mechanisms are almost adapting themselves to new threats and attacks. Probably the best example are the motif-based attack fingerprints, that are calculated for every new attack. The unsupervised learning (cf. Section 5.3.2.3) can then even create new attack signatures when a previously unknown attack scenario is observed. Also the alert clustering in GAC with its flexible definition of similarity for the attributes of alerts, requires no particular domain knowledge to run.

Resilience and Self-protection

The distributed deployment of the zeek-osquery platform can be used to monitor several parts of the network with multiple Zeek instances and to perform intrusion detection for each network part locally as much as possible before correlating the detection results (cf. Section 5.3.2). This adds resilience to the overall detection system as an attacker might achieve to disable the detection mechanisms for parts of the network. However, it is unlikely that the attacker achieves this for the whole network altogether.

A concrete attempt of the attacker to effectively disable the detection is to overwhelm the detection system with artificially generated events and noise, and thereby obfuscating the original attack. While this indeed could cause some detectors and alert correlations to be overloaded, there is likely a stage in the processing pipeline that can still cope with the extraordinary amount of data or that can at least log the traces of the attack at this particular processing stage (cf. Section 4.1.3).

Privacy

Privacy can be seen as a matter of external concerns with respect to third parties gaining access to internal information but as a matter of sensitive information being disclosed to the detection system itself and the security operators. On these two levels, privacy concerns have been identified while designing the detection system proposed in this thesis.

With respect to data collection, some information such as the network traffic are argued to be mandatory to perform intrusion detection accurately. Similarly, some information about the hosts such as processes are required for attributing network flows (cf. Section 4.1.1) to host applications. However, some other especially more sensitive information of the hosts like file content or Transport Layer Security (TLS) session keys are also extremely helpful for intrusion detection. To value the privacy of the users, zeek-osquery allows for interactive data retrieval. This way, sensitive information is only transferred to the detector if there is an explicit situation demanding this information. Furthermore, it is not the actual file content that is collected but the hash of the file.

On the level of alert correlation, a privacy concern arises when sharing data in a CIDS. The mechanism to exchange alert summaries is based on the attack scenario and excludes any concrete alert attribute values such as IP addresses or ports (cf. Section 5.3.2). The result is that the attack itself is fingerprinted to produce the alert summary for exchanging, but it does not contain any sensitive information about users or the involved systems.

The next sections of this chapter answer all remaining questions from Section 1.1 in an quantitative evaluation.

6.3 Evaluation of Joint Monitoring with zeek-osquery

The probably most common deployment of security monitoring is a network-based IDS that captures and analyses the network traffic of the hosts in the network. Network monitoring, however, is giving no insight into the hosts, and the trend towards encrypted communication additionally impedes the detection with a NIDS. zeek-osquery (cf. Section 4.1) refines the

visibility of security monitoring to overcome these challenges. The key idea is to correlate network data with additional monitoring data from hosts in real-time. In particular, network flows are attributed to the application and user on the respective host.

This evaluation of zeek-osquery investigates how security monitoring can benefit from the joint host and network monitoring. Apart from enhancing the detection accuracy, also other requirements from Section 3.1 must be covered by the zeek-osquery system to be applicable in the real-world. In particular, the system must be highly efficient and scalable to process the large data volume that results from closely monitoring large networks.

This section revises parts of the conference paper [HSF20]. The section evaluates the performance of security monitoring with zeek-osquery for a joint monitoring. First, a real-world deployment gives insights into the benefits of extending the visibility by host data. Afterwards, the open-source prototype is stress-tested with more hosts and host events to evaluate its scalability properties. Last, a testbed with actual attacks demonstrates the benefits of the joint monitoring regarding the enhancement of the detection accuracy.

6.3.1 Real-World Evaluation

The monitoring accuracy of zeek-osquery is determined by the success rate of correlating host and network data. The most preliminary correlation is the attribution of network flows to the respective application and user (cf. Section 4.1.1.2). Thus, the following real-world evaluation in this section assesses zeek-osquery regarding identifying the application and user of a network flow in real-time (cf. Section 4.1.3).

6.3.1.1 Evaluation Setup and Dataset

A working group of the computer science department deployed zeek-osquery on the university campus to evaluate the accuracy of zeek-osquery under real-world conditions. Eleven employees participated in this experiment by monitoring their eleven office machines with zeek-osquery for three working days. The machines were running different Linux distributions, including Ubuntu, Linux Mint, Fedora, and Arch Linux. To monitor their network traffic and to correlate it with host events, the participating users tunneled their traffic through a virtual private network (VPN) on the campus site that is monitored by a Zeek instance. Note that the correlation is performed on live traffic and also in real-time during this experiment. The users have been aware of the monitoring mechanism in place while being connected to the VPN. They could leave it at any time to pause the monitoring. Knowing to participate in such a monitoring experiment potentially influenced their online behavior. However, measuring the technical accuracy in this experiment only little depends on the exact online behavior of the users. Instead, the intention is to have a realistic and broad setup with real-world applications and traffic.

The data processed by zeek-osquery in this setup is characterized in Table 6.3. It reports characteristics of the flows that zeek-osquery ideally should correlate with host data. Not all recorded data flows made it into this dataset because of two reasons. First, unsolicited messages to closed ports on monitored hosts cannot be attributed to any process in the first place. Second, some users joined the VPN with their hosts but started osquery later. Network flows from both situations were filtered from processing.

Network flows				Host events for state			
Total	TCP	UDP	ICMP	Process	Socket	User	Interface
344,366	273,241	70,929	196	2,793,406	776,910	51,719	7,919

Table 6.3: Characteristics of the real-world dataset.

Furthermore, the table reports the number of received host events. On average, each of the eleven machines was monitored for 6 hours and 21 minutes per day. An individual host on average reported 222.4 process, 20.6 socket, and 4.1 user events per minute that go into state (cf. Section 4.1.1.3). These events sum up to 4.1 events per host and second on average. Note that also the initial state that is retrieved from hosts upon their (re-)connects goes into the average event rate. Thus, many initial events are retrieved every time a host joins the VPN, which happened several times as some users decided to leave the VPN for short periods during the day.

The following experiments provide results and experiences on how zeek-osquery enhances the visibility and accuracy of monitoring.

6.3.1.2 Attribution of Network Flows

To assess to which extent zeek-osquery can attribute network flows to host processes in real-time, this experiment analyses the attribution result of the real-world deployment. Table 6.4a shows the success rate of zeek-osquery attributing the 344,366 network flows in the dataset. For 96.05% of Transmission Control Protocol (TCP) connections and 86.61% of all flows, the responsible processes and users are identified. False negatives are caused when: First, the host data is not retrieved in time for real-time correlation with short-lived flows. Second, applications like `Skype` use Stateless IP/ICMP Translation (SIIT) to embed the actual IPv4 destination address into an IPv6 address. However, as the hosts in this experiment are configured with IPv4 only, they send out the message to the embedded IPv4 address. Consequently, this causes a mismatch between IPv4 (in the network flow) and IPv6 addresses (in host events). Third, remote hosts sometimes try to continue a flow, although the monitored host already left the VPN. When in the meantime, a new host joined the VPN reusing the same IP address these packets cannot be attributed to a process on the new host. The attribution rate for User Datagram Protocol (UDP) flows is only about 50% because Zeek retrieves host events about UDP sockets from the audit status only (cf. Section 4.1.1.1), which is provided at discrete time slots only. Thus, short-living sockets might be missed out. This holds especially true for Domain Name System (DNS) requests that are responsible for 89% of the UDP flows.

All	UDP	TCP
86.61%	50.43%	96.05%

(a) Attribution rate.

	Host	Process	User
Unique attributions	100%	88.53%	98.14%
Average candidates	1.00	1.17	1.02

(b) Attribution uniqueness.

Table 6.4: Attributing of network flows.

This experiment further evaluates the attributed flows with respect to a unique host, process, and user. Table 6.4b counts the number of flows that have been attributed to a single entity and

	Zeek		zeek-osquery	
Rank	Attributed flows:	0.06%	Attributed flows:	86.61%
1	Chrome	(0.01%)	Firefox	(23.17%)
2	Firefox	(0.01%)	Thunderbird	(12.30%)
3	Spotify	(0.01%)	Spotify	(6.11%)
4	Thunderbird	(0.01%)	Opera	(5.41%)
5	Debian APT-HTTP	(<0.01%)	Syncthing	(5.39%)
6	libdnf	(<0.01%)	Chromium	(4.55%)
7	Wget	(<0.01%)	Skype	(3.87%)
8	<unknown browser>	(<0.01%)	Seafile	(3.80%)
9	OpenSSH	(<0.01%)	Chrome	(3.56%)
10	gvfs	(<0.01%)	qutebrowser	(3.33%)
Total	33 applications		88 applications	

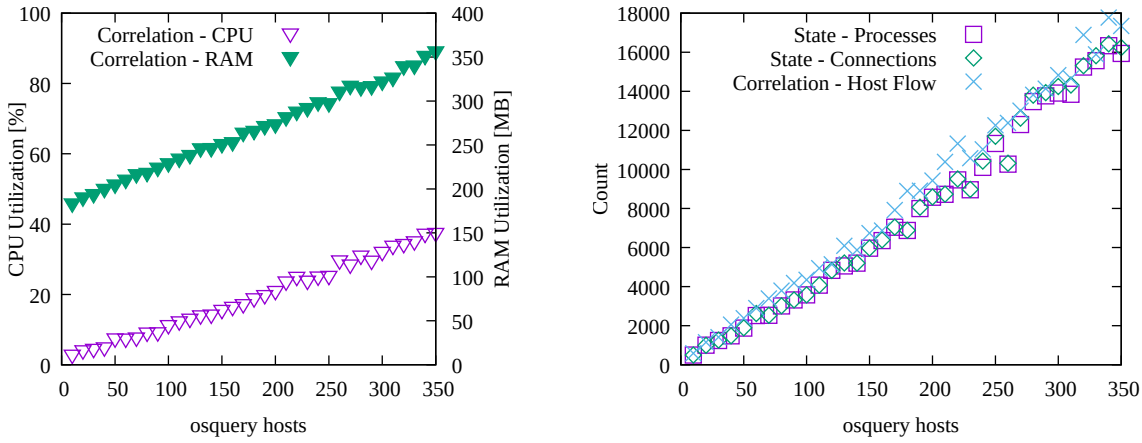
Table 6.5: Top 10 attributed applications among all network flows.

furthermore calculates the average number of candidate entries per attribution. Apart from the vague correlation (cf. Section 4.1.1.2), a fast re-usage on hosts of the same process ID or socket, i.e., file descriptor, can be a reason for multiple attribution candidates. The effects of the vague correlation become visible, especially for DNS flows. Usually, applications use the DNS server defined by the OS, and therefore many processes establish flows to the same server and port combinations. When skipping the attribution of flows to the DNS servers, the unique attribution of processes increases from 88.53% to 93.15%. Although a single user was logged in on the monitored machines, in some cases, the user attribution overlaps with a system account, e.g., in case of parallel DNS requests by the system and a user application.

6.3.1.3 Identification of Host Applications

For identifying communicating applications, the state of the art is to inspect network packets for application-specific indicators like the Hypertext Transfer Protocol (HTTP) user agent. Zeek already analyses such indicators and derives the respective application, where applicable. If Zeek itself cannot derive the application from the network packets, zeek-osquery can still verify the application via the correlation with host data. Table 6.5 lists the top 10 network applications ranked by their number of attributed flows. Two outcomes are observed when comparing both methods for identifying communicating applications: First, zeek-osquery is able to attribute significantly more flows compared to Zeek, i.e., 298255 (86.61%) compared to 212 (0.06%). For the Firefox browser, zeek-osquery was even able to attribute flows 2971 times more often than Zeek. Second, zeek-osquery can identify applications that were not identified by Zeek. This includes user applications such as `Syncthing`, `Seafile`, and `Skype`, but also system-related components such as the network time synchronization daemon `NTPD` and the Dynamic Host Configuration Protocol (DHCP) client `dhclient`.

However, also limitations of zeek-osquery have been seen in this experiment, especially when a process launches another application that immediately starts a network flow. Because the flow could have happened before or after the parent process with the same pid transferred control with the `execve` syscall to the new child, both parent and child application are candidates for the attribution. In this experiment, applications that are known never to communicate directly are candidates for 0.18% of attributed flows. Also, some monitored hosts were running virtual



(a) CPU and RAM utilization.

(b) State and correlation statistics.

Figure 6.2: Zeek performance when hosts continuously report four host events per second.

machines (VMs) with a Windows guest system behind network address translation (NAT). While osquery runs on the Linux hypervisor host only, it attributes any traffic of the VM to the virtualization application (2.29%). However, Zeek might still identify the Windows application inside the VM based on identifiers in the network packets (0.01%).

In summary, zeek-osquery significantly increases the identification rate of communicating applications. This enables to enforce the use of allowed applications and assists threat hunters in detecting malware that covers its communication in well-known protocols, e.g., Hypertext Transfer Protocol Secure (HTTPS), that is usually allowed to pass the firewall.

6.3.2 Performance Analysis

Apart from working accurately in general, zeek-osquery also has to do so not only for a small network with few hosts but for large networks as well. In particular, the real-time requirement must still be fulfilled even a large amount of data from many hosts is processed. Thus, the following performance analysis assesses the scalability and efficiency of zeek-osquery with an increasing number of osquery hosts and an increasing number of events. For that, two experiments are conducted to measure the overhead of Zeek and osquery, respectively.

6.3.2.1 Zeek Overhead

The first experiment for performance analysis investigates the load on Zeek in means of CPU and RAM for handling osquery hosts and processing host events. For that, multiple osquery hosts establish a connection to Zeek and regularly send updates about local processes and connections. Zeek then correlates these events to link processes to their sockets, and holds this mapping as long as the process and connection stay alive (cf. Section 4.1.3). This mapping effectively implements a state of flows by processes, which is denoted as *host flow* in the following.

As the Zeek overhead is the scope of this measurement, no full-fledged osquery hosts are required, but rather a controllable environment to put load on Zeek. For this reason, the experiment utilizes

a simplified Python prototype of the actual Zeek-enhanced osquery implementation. With the same interfaces exposed like the actual implementation, the simplified prototype can be queried by Zeek for some simple events processed during the experiment. The prototype simulates a host to start short-lived processes and connections constantly. Each host is configured to continuously send four events per second to Zeek, which is almost exactly the same as in the real-world evaluation (cf. Section 6.3.1.1). This lightweight implementation also enables us to simulate a lot of osquery hosts that run on a few physical machines only.

The total amount of the lightweight osquery instances is equally distributed among ten bare-metal machines. Zeek takes over the role of both proxy and authoritative Zeek and is running on another bare-metal machine. This comes close to a real-world deployment, in which Zeek runs on a single machine, and osquery instances are distributed on different machines in the network. All of these machines are equipped with an Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz and 8GB of RAM. Anyhow, the performance of Zeek is measured for a specific number of hosts or host events per second, respectively. Each configuration setup runs for 20 minutes while measuring resource utilization and correlation statistics every second. The following plots report the average measurement value over the whole 20 minutes for each configuration.

Figure 6.2a plots the average CPU and RAM utilization of the Zeek host in dependence on the number of connected osquery hosts. During the experiment, Zeek retrieves and logs host events about processes and sockets, correlates them to host flows, and reconstructs their state as it is required for the attribution of network flows. The result indicates that the resources scale linearly with an increasing number of osquery hosts. A single host causes 0.11% CPU and 0.45MB RAM at the Zeek instance during real-time correlation. Theoretically, to achieve 100% CPU utilization, about 870 osquery hosts would be required, each sending four events per second.

Figure 6.2b plots the average number of state entries for processes and connections, as well as the average number of correlated host flows at the Zeek instance in dependence on the number of osquery hosts. This result also indicates a linear dependency on the number of hosts for the size of state and correlation. Because the simplified osquery hosts simulate an equal amount of processes and connections, the three curves are expected to be identical. The observed unsteadiness in the curves is caused by simulation randomness for the event generation that sometimes leads to fast re-usage of process IDs or file descriptors in the process and socket events, respectively. Thus, this experiment confirms that the state reconstruction in the zeek-osquery processing pipeline works correctly.

6.3.2.2 osquery Overhead

The second experiment for the performance analysis investigates the load on hosts when they run osquery and, in particular, as part of a zeek-osquery deployment. Load on hosts manifests itself in CPU and RAM load caused by osquery, mainly to execute the SQL queries and to log query results. The modifications to osquery mainly affect its logging, as osquery sends query results to Zeek via Broker. Thus, this experiment abstracts the load to run osquery by simplifying the execution of SQL queries. For that, the implementation of a dummy table in osquery with three columns contains a predefined number of results, which is queried by Zeek. This static table is queried in combination with different logging mechanisms to compare their overhead, including the mechanism via Broker.

For this experiment, the modified osquery is deployed on a desktop machine running Ubuntu 18.04 equipped with an Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz, 8GB RAM, and 256GB SSD. The same SQL query for the static dummy table executes every ten seconds, which each time returns 100,000 results. Table 6.6 illustrated the measurement results for the average CPU and RAM utilization over 90 seconds for the built-in osquery logging mechanism that can either log to a logfile or to a remote web server via TLS[2]. These results are compared with the Broker logging provided by the zeek-osquery modifications and with the baseline when osquery logging itself is turned off.

	Loss [%]	CPU [%]	RAM [MB]
No Logging	-	2.67	615
Logfile	0	6.44	1108
TLS	95	5.29	1402
Zeek	0	7.83	790

Table 6.6: Performance comparison of the different logging mechanisms.

First of all, the table verifies the correctness of logging by counting the logged results and comparing them to the expected number of results. A correct logging is seen when using a local logfile and also using Zeek as part of a zeek-osquery deployment. For TLS, however, it is indicated that the logging of so many events is not possible, e.g., the status logs of osquery report the log buffer to overflow. This leads to a result loss of 95% in this experiment. The reason might be that the request-response pattern in HTTP causes to much delay to transfer a large amount of results and therefore causes congestion at the sender.

Furthermore, the experiment for the osquery performance indicates that the remote logging to Zeek via Broker promises a reliable way to send query results. It can process large amounts of query results without stressing the osquery hosts significantly. Compared to default logging to a logfile, the modified version requires only 1.4% more CPU and even uses 200MB less RAM.

6.3.2.3 Performance Discussion

The performance experiments indicate that zeek-osquery is an efficient platform to collect host and network events. This efficiency is enabled by the zeek-osquery design to retrieve only raw events from osquery hosts and not to let the hosts themselves perform resource-intensive analysis. This way, the overhead on osquery hosts is kept small. The actual load on Zeek depends on the amount of host events and the kind of correlations that it has to perform. For rudimentary correlation purposes, a single Zeek instance might be suitable for a network of small to medium size. If a single Zeek instance cannot cope with the load or if complex correlations are required, a more scalable distributed deployment can be used, as described in Section 4.1.2.3.

6.3.3 Scenario Evaluation

The joint host and network monitoring with zeek-osquery is not only intended for better monitoring but explicitly for better intrusion detection, i.e., higher detection accuracy, as well. Thus,

2. Using the reference implementation mentioned in:
https://osquery.readthedocs.io/en/stable/deployment/remote/#server-testing

the following evaluates the implementation of two scenarios from Section 4.1.4 in Zeek scripts, namely the execution of Internet files and Secure Shell (SSH) hopping.

6.3.3.1 Deployment

The experiment uses a testbed deployment to evaluate the detection of specific scenarios with zeek-osquery. The testbed contains Debian 10 VMs and scripts real applications to communicate on the local network and the Internet, respectively. In particular, the experiment runs the Firefox web browser to display random websites from the Alexa top million sites. Furthermore, the experiment includes an email server in the testbed and runs the Thunderbird mail client to send and receive emails with attachments. Furthermore, the Linux SSH tools are used within the experiment to randomly login to other VMs in the testbed.

The testbed architecture looks as follows (cf. Figure 6.3): A local network with ten VMs is monitored by zeek-osquery. Every VM runs an osquery instance, and Zeek captures both their internal and external communication. Each VM is continuously surfing the web, writing emails including attachments, and establishing SSH sessions to other VMs. For that, every VM is configured with random periods between performing network actions using real applications. Firefox uses about 5 to 50 tabs to display the start page of random websites on the Internet, including all embedded content. Thunderbird is deployed with different mail accounts on each VM, configured to use Post Office Protocol version 3 (POP3) for their inboxes. User activity is simulated to write emails to each other using plain Simple Mail Transfer Protocol (SMTP) continuously, and the emails might contain random attachments. Additionally, the VMs randomly establish SSH connections among each other for short periods.

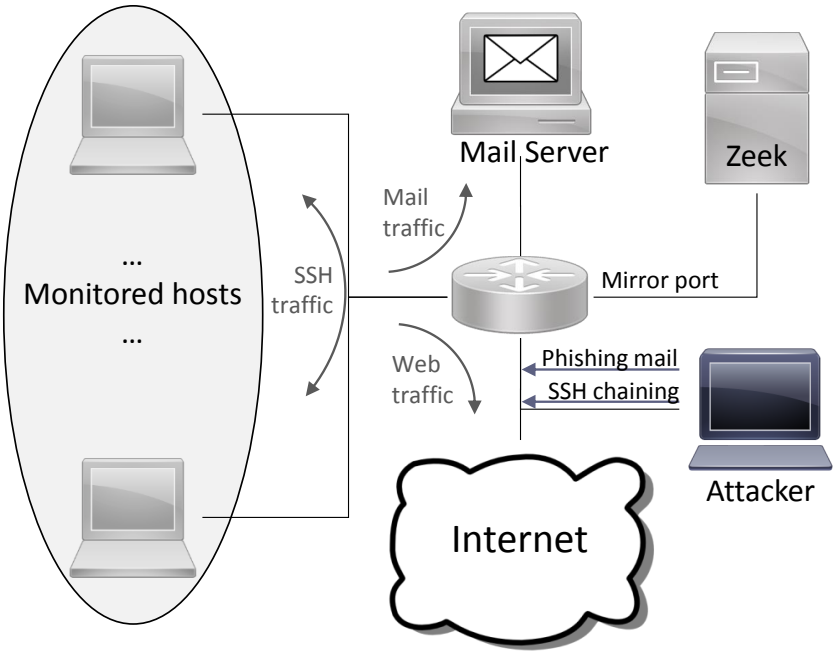


Figure 6.3: Architecture of the testbed.

The testbed contains an additional attacker machine that (1) sends malicious mail attachments, and (2) abuses the VMs as SSH jump hosts.

6.3.3.2 Findings

During a runtime of six days, Zeek was analyzing over 5.61 million network flows. This amount of network flows corresponds to the behavior of about 2.37 always-on hosts compared to the real-world evaluation. Across the deployed VMs, web browsing caused 2.37 million connections, sending mails caused 8,420 connections, and receiving them caused 34,560 connections. Furthermore, 62,109 SSH connections were seen in the network. The high number of connections is on purpose, as the application scripting is intended to make extreme use of network communication to generate a high volume of benign background traffic. To detect the few attacks within this traffic can be considered the worst case.

During the experiment, the attacker performed each attack once. It sends a crafted mail, including a potentially malicious executable to one of the mail accounts. As with all other received emails, the respective user downloads the mail and attachment. For this specific email, the attachment is executed after download. To perform SSH hopping, the attacker logs in to one of the VMs and abuses this SSH connection to tunnel another SSH connection towards another VM in the testbed.

The log files for the specific scenarios show that zeek-osquery detected the two attacks correctly. Furthermore, none of the other 3,892 attachments triggered an alarm, nor was any other SSH connection flagged as SSH hopping. For the evaluated scenarios, the detection accuracy is 100%, as there are only true positives but no false positives or false negatives. Although this perfect accuracy might not be directly applicable to other intrusion scenarios, zeek-osquery demonstrates the benefits of causally correlating the data from host and network monitoring.

6.3.4 Summary of Joint Monitoring

An evaluation of zeek-osquery has been presented for the mitigation of NIDSes going blind. The open-source zeek-osquery system compensates restricted visibility into the network traffic by complementing data from the hosts about their network communication. Through a fine-grained correlation of the data, zeek-osquery is able to attribute network connections to users and applications in real-time. On top, custom detection scripts can leverage the resulting data in a correlation framework for intrusion detection or threat-hunting.

The zeek-osquery prototype has been deployed in a real-world evaluation for insights into the data correlation and the resulting visibility. In this experiment, zeek-osquery was able to attribute over 96% of the TCP connections in real-time. For that, hosts sent on average 4.1 host events per second to Zeek. While a NIDS can usually just derive the type of application from the communication protocol, the attribution with zeek-osquery reveals the concrete application on the host. In contrast to a network-based identification of applications, zeek-osquery identifies more than double the amount of unique applications. While this is a smaller improvement, the benefit becomes apparent with the number of flows for which an application was identified: It is improved by factor 1406.

Furthermore, the zeek-osquery prototype has been stress-tested to investigate its scalability and efficiency. The resources of a central Zeek instance scales linearly with the number of monitored osquery hosts. While a single instance can eventually monitor up to 870 hosts, a distributed deployment allows to scale with an arbitrary number of hosts. Apart from the overhead of running osquery itself on the host in the first place, the implemented communication channel

to Zeek for receiving SQL requests and sending host events might cause additional overhead. The experiments measured no significant overhead when a host running osquery is becoming part of a zeek-osquery deployment. It even performs stable under high load, while other remote logging mechanisms fail.

Last, the new detection capabilities with zeek-osquery have been demonstrated in a testbed, in which hosts were running real-world network applications. Custom detection scripts leverage the correlated host and network data from joint monitoring to detect the execution of a malicious mail attachment and to detect an SSH stepping-stone attack.

In summary, zeek-osquery enables a fine-grained monitoring of the network and the hosts. The evaluation of the open-source prototype highlighted both the benefits of the refined visibility for security monitoring and the practical aspects for a real-world deployment.

Note that zeek-osquery requires to deploy the osquery sensor on the hosts. While applicable to certain environments where the hosts run under the authority of the network operator, administrative access to the hosts is not always possible in other environments. Thus, the next section evaluates two approaches for security monitoring that work without any help of the hosts.

6.4 Evaluation of Correlated Network Communication for Intrusion Detection

Apart from an in-depth security monitoring with heterogeneous data sources (cf. Sections 4.1 and 6.3), the security monitoring and detection accuracy can benefit from a holistic assessing of network traffic. For that, an attack scenario is detected across relevant events by assembling them to a bigger picture. Such kind of scenario detection works especially for network-wide attacks such as distributed attacks, e.g., scan campaigns (cf. Section 4.2) and P2P botnets (cf. Section 4.3).

Instead of assessing events of an attack scenario individually, the collection of relevant events enables the scenario detection. For such an intrusion detection based on correlated events, the amount of monitored activities is highly relevant. The detection is assumed more accurate, the broader and sound the visibility on the attack is. Thus, the highest accuracy is assumed when all activity related to the attack is monitored and available as events.

However, network monitoring is usually limited to the boundaries of the own network. In case of attacks with Internet scale, regular network monitoring cannot capture the full scenario. Thus, this evaluation investigates how accurately such Internet-wide attacks can be detected with the respective detection algorithms, despite a limited visibility in network monitoring.

This section revises parts of the conference papers [HWF20] and [Muh+18]. The section evaluates the detection accuracy of two scenario-specific detection algorithms regarding limited visibility. First, the detection results of scan campaigns is reported and compared for different network sizes. Second, the evaluation of the P2P botnet detection investigates the required visibility for an accurate detection.

6.4.1 Scan Campaign Detection on the Internet

The evaluation of detecting port scan campaigns (cf. Section 4.2) performs on real-world Internet traffic. First, a characterization of the respective data set is given. After analyzing the detection algorithm in a parameter study, some scan campaigns in the detection results are highlighted. These detected campaigns are compared across the perspective of the backbone infrastructure, an Internet service provider (ISP), and an enterprise, to discuss their detection accuracy.

6.4.1.1 Visibility on Port Scans

The evaluation setup utilizes a large trace of Internet traffic that is considered as ground truth. Based on this trace, restricted network visibility is simulated to reflect the scope for different levels of network providers. Both the original data set and the simulated restriction are detailed next.

Evaluation Data and Tools The traffic trace comes from the MAWILab project [Fon+10] that monitors a transit cable between the USA and Japan. For anonymity reasons, only IP and TCP/UDP headers are recorded in this large trace. In addition, the last byte of an IP address is randomized consistently among the data set. This potentially effects the key features for the IP and geolocation similarity of the detection algorithm. However, as the address modification is at the last byte only, the distance between two IPs changes only marginally and the geolocation stays most likely unchanged.

For the detection of scan probes, the network monitor and IDS *Zeek* [Pax99] is used. It analyzes the network trace for TCP port scans, i.e., incomplete TCP connection establishments. For that, Zeek implements a connection tracking that identifies an incomplete TCP three-way handshake [Pos+81], e.g., because the destination port is closed or the connection remained half opened. Apart from rare technical reasons like misconfigurations, this usually indicates an attacker that is only interested in discovering open ports, but not in establishing a full connection. In the MAWILab capture of 15 minutes on May, 5th 2019, Zeek detects $9,960,652$ scan probes from $199,403$ unique source IP addresses, i.e., scanners, to $265,081$ unique destination IP addresses, i.e., targets.

Scope	Subnet	Scan Probes	Scanners
Backbone	0.0.0.0/0	9,960,652	199,403
ISP	133.242.0.0/16	2,228,873	67,929
Enterprise	133.242.179.0/24	13,598	1,070

Table 6.7: Port Scan statistics for full data set and simulated network operators.

Restricting Network Visibility The MAWILab capture and therefore the detection of the port scans are in the scope of an Internet backbone, as the traffic was recorded from a transit cable. This reflects the ground truth with all the host communication in the data set. Hence, the full data set is the benchmark comparison when it comes to the detection of scan campaigns from the perspective of targeted networks. In contrast to the holistic view on the scan campaign from the perspective of the Internet backbone, a targeted network might only be one of many targets

in the campaign. Therefore, the targeted network eventually has only a limited view on the campaign, especially when the campaign is Internet-wide.

To simulate smaller networks, the visibility of the monitoring and scan correlation is restricted to a fraction of IP addresses, i.e., an IP subnet in the full data set. Consequently, the restricted data sets contain only incoming and outgoing communication with respect to the monitored subnet. Apart from the original data set as backbone scope, the restriction simulates the scope of an ISP and an enterprise. For the respective subnets, an IP subnet is chosen that received a lot of scans. The resulting three data sets of different scopes are summarized in Table 6.7.

6.4.1.2 Parameter Study

This first evaluation part discusses the most influencing parameters for the detection of port scan campaigns. In this parameter study, the performance of the scan correlation on the backbone scope is investigated in dependence on individual parameters.

False Positive Filter First, the threshold ε to filter false positive scan alerts is analyzed. This parameter defines the minimum number of required scan probes per scanner. Together with the scanners, their scan probes are filtered as well. Analyzing this parameter gives insight into how large the portion of filtered scanners and scan probes is. Thus, this analysis applies different values for ε and illustrates the portion of remaining scanners and scan probes in Figure 6.4.

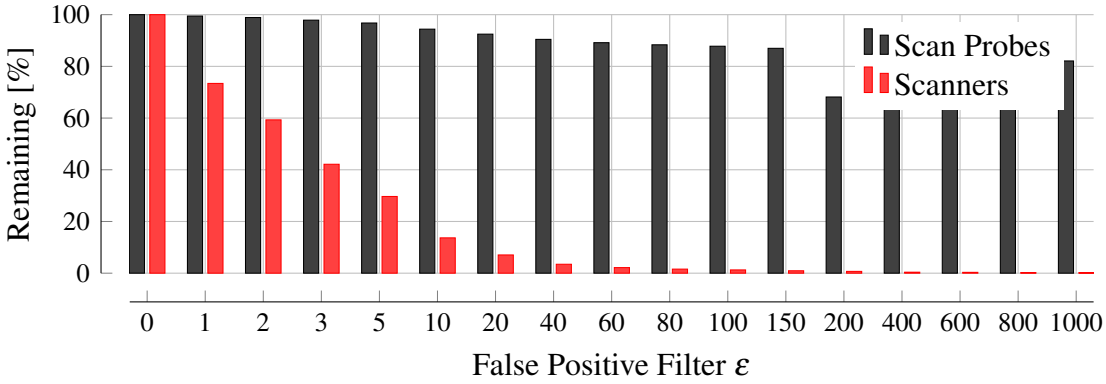
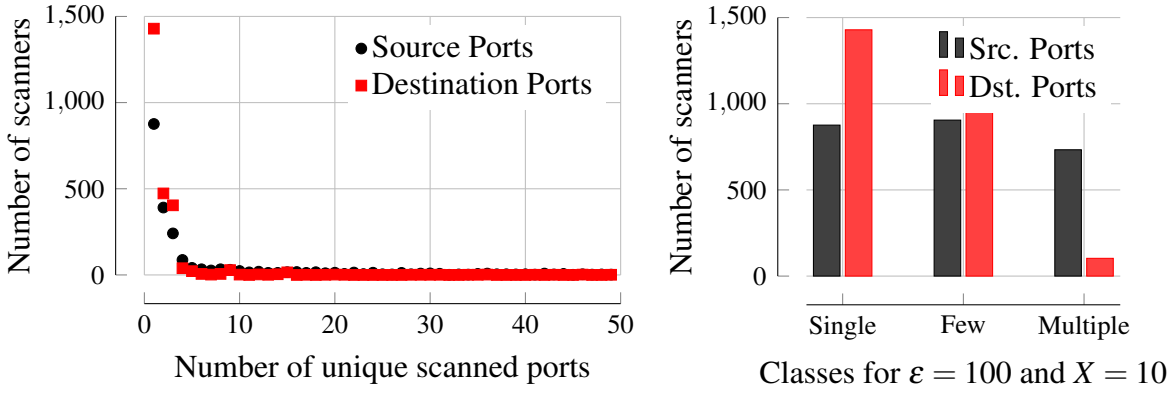


Figure 6.4: Filtering effect of the false positive filter ε on the portion of remaining scan probes and scanners.

The plot shows a clear difference between remaining scanners and probes. The portion of remaining probes is not in line with the portion of remaining scanners. Consequently, there is an imbalance in the distribution of probes among the scanners. On average, each scanner originates 50 probes. However, less than 3% of the scanners remain for $\varepsilon = 50$. At the same time, more than 90% of the probes remain. Furthermore, about 90% of the scanners cause 15 or less probes. Consequently, there exists a few scanners that cause hundreds of probes. For example, for $\varepsilon = 5$ only 30% of the scanners are left but they still count for 96% of scan probes. For $\varepsilon = 100$ even the remaining 1.26% scanners count for 88% of probes.

Sources with only a few probes are unlikely to be scanners. Instead of scan behavior, it is more likely caused by technical failures that regularly occur on the Internet. The analysis of ε indicates, that a large portion of false positive scanners can be filtered with a small ε.

Port Classes The scans of each scanner can target different ports across the targeted hosts. This analysis of port classes depends on the parameter X that defines the number of unique ports in a scan as the threshold between the classes $\mathscr{F}$ew and $\mathscr{M}$any. Figure 6.5a plots the distribution of unique ports, both source and destination ports, among the scans to decide on the parameter X. The plots illustrates how many scans with a particular number of unique ports exists in the real-world data set. It looks close to a exponential distribution. In further experiments, the threshold is set to $X = 10$, as for the shape for the distribution function changes around this point.



(a) Number of ports.

(b) Classes of ports.

Figure 6.5: Distribution for number of unique scanned ports among the attackers.

Figure 6.5b illustrates the resulting classes for ports when applying the parameters $\varepsilon = 100$ and $X = 10$. For this parameter setting, the most scans are labeled with $\mathscr{F}$ew source ports and a $\mathscr{S}$ingle destination port.

Similarity Weights The similarity weights $w(i)$ are designed to prioritize particular similar features i. Prioritized features have a higher influence in the similarity calculation when it comes to identifying related port scans. For the following experiments, the feature weights are categorized into three groups with different weights. The strongest weight of 4 is for the two ports. Medium strength with a weight of 2 is assigned to the vertical and horizontal property, scan verification, and source IP and location. The lowest weight of 1 is assigned to the IP version, and the order of magnitude for target hosts and scan probes.

Clustering Cutoff The effect of the parameter t for the cutoff in hierarchical clustering is illustrated in Figure 6.6. For the following experiments, the clustering cutoff is set to $t = 0.15$ as a reasonable value, which is in between two points where the shape significantly changes.

After this parameter study, the detection parameters are $X = 10$, ε between 0 and 10, and $t = 0.15$. Next, the collaboration evaluation analyzes the resulting clusters, i.e., scan campaigns, for different scopes of network visibility.

6.4.1.3 Collaboration Evaluation

Applying the correlation of port scans with the parameter values from the parameter study on the evaluation data set, results in the campaigns summarized in Table 6.8. As the simulated scope of

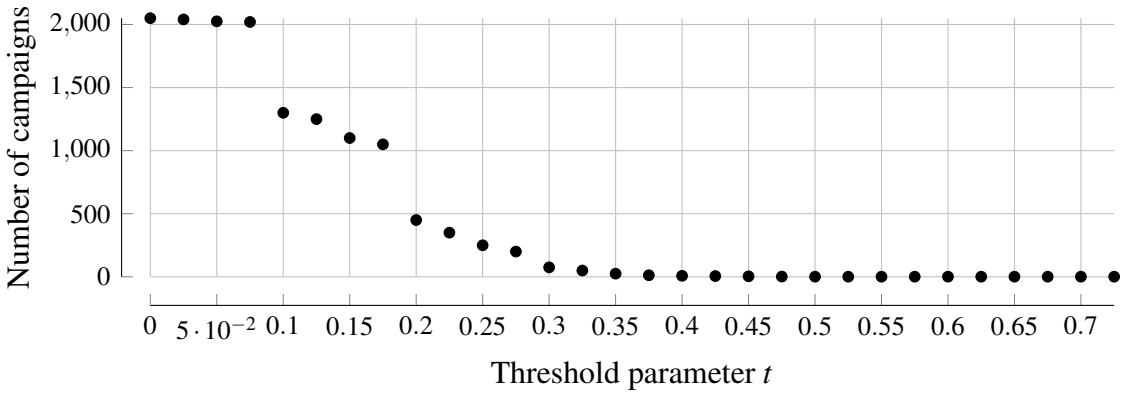


Figure 6.6: Resulting clusters depending on the parameter t.

an ISP and enterprise network restricts the visibility, the number of scanners decreases to 32% and 4%, respectively, compared to the ground truth in the backbone scope. Consequently, also the number of detected campaigns decreases to 9% and 1.6%, respectively.

Scope	Filter ε	Scan Probes	Scanners	Campaigns	Distributed Scanners		Accuracy
Backbone	10	9,401,543	27,244	1955	19,373	(71%)	100.0%
ISP	5	2,130,215	8,617	717	5789	(67%)	94.5%
Enterprise	0	13,598	1070	32	208	(20%)	27.3%

Table 6.8: Summary of campaign detection for different visibility scopes.

Discussing the Detection Accuracy The decrease of scan probes, scanners, and campaigns is directly caused by the restricted visibility scope. Beyond that, this experiment investigates the consequences of this decreased on the accuracy of correlating monitored scans to campaigns. Table 6.8 also lists the number of distributed scanners or the fraction of scanners that are identified to collaborate in a campaign, respectively. Similar to the overall number of scanners, also the absolute number of distributed scanners decreases with restricted visibility.

The relative number of distributed scanners also decreases with restricted visibility from 71% at backbone scope to 20% at enterprise scope. Assuming a constant fraction of distributed scanners at any scope, the 71% distributed scanners in the ground truth of the backbone reflects 100% accuracy. Consequently, this then indicates an accuracy of 94.5% at ISP scope and 27.3% at enterprise scope compared to backbone scope.

However, this accuracy calculation ignores the fact that scan campaigns might sample the global IP space. With a smaller network scope, also the probability to be targeted by two coordinated scanners decreases. Thus, the small enterprise network eventually cannot monitor scan probes of different sources that belong to the same campaign in the first place.

Highlights in Comparing Campaign Characteristics Out of the 1955 detected campaigns on backbone scope, two specific ones are characterized in detailed here. They represent many other campaigns that have been found in the data set. Even though, they exemplary highlight the clustering outcome as summary of a scan campaign. In particular, outcome of the same two

campaigns is compared on all three visibility scopes. The fingerprints of the campaigns are shown in Tables 6.9 and 6.10. Apart from the number of scan probes, the tables list the values of the ten key features (cf. Table 4.2). The tables illustrate feature values that are equal for all visibility scopes and others that differ among the scopes.

	Enterprise	ISP	Backbone
Scan Probes	20	4,479	18,259
Source Ports	$\mathscr{S}+30443$		
Destination Ports	$\mathscr{S}+30443$		
Vertical Scan	*true* and *false*	*true*	
Horizontal Scan	*false*		
Scan Validation	*false*		
IP version	*IPv*4		
Target Hosts	1 to 3	145 to 196	591 to 799
Port Scans	1 to 3	145 to 196	591 to 799
Source Subnet	15 addresses in 88.138.143.0/27	27	27
Source Location	France		

Table 6.9: Summary of a campaign from France for different network scopes.

A scan campaign from France is summarized in Table 6.9. The key features show that many values are equal among the distributed scanners. In this campaign, 27 out of potentially 32 IP addresses in the 88.138.143.0/27 source subnet show the same scanning properties. This is a very strong indication for their coordination. While these 27 scanners are clustered at backbone and ISP scope, the monitoring at enterprise scope captures only 15 of these scanners. However, they are still clustered at this scope. Apart from the source subnet, the table reveals some other common scan features.

	Enterprise	ISP	Backbone
Scan Probes	2449	450,489	1,812,160
Source Ports	$\mathscr{S}+46960$ and $\mathscr{S}+55776$		
Destination Ports	$\mathscr{M}$		
Vertical Scan	*true*		
Horizontal Scan	*true*		
Scan Validation	*false*		
IP version	*IPv*4		
Target Hosts	256 and 248	65,205 and 53,011	260,299 and 211,552
Port Scans	1747 and 702	343,609 and 106,880	1,382,844 and 429,316
Source Subnet	2 addresses in 185.173.217.208/28		
Source Location	Netherlands		

Table 6.10: Summary of a campaign from Netherlands for different network scopes.

In the Netherlands campaign in Table 6.10, only two scanners are clustered. These two scanners are in the same /28 subnet. That they use two different source ports might not be intuitive at a first sight. However, that they both use a single source port is already a common characteristic. Most likely, both scanners deploy the same scan tool that uses a static port that is not hard-coded but chosen at run-time.

For the demonstrated detection results of scan campaigns, the common features among the scanners were highlighted. To some extent, the scanners can be correlated even at smaller visibility scopes. Generally, the detection is likely to be successful as long as a sufficiently large number of scan activities from a particular campaign is monitored.

6.4.2 Botnet Detection on the Internet

The evaluation of detecting P2P botnet (cf. Section 4.3) performs on real-world network and botnet traffic. First, a characterization of the respective data sets is given. Afterwards, the detection accuracy under two visibility restrictions is evaluated to discuss the required visibility in network monitoring to detect P2P botnets on the Internet.

6.4.2.1 Ground Truth in Communication Graphs

Evaluating the detection algorithm ideally requires labeled real-world data. However, global NetFlow records for real-world botnet infections that cover all bot communication with ground truth is not available. Instead of a real-world communication graph G_N with ground truth, this graph is constructed from a background graph G_L and a botnet graph G_B following the model in Section 4.3.1. For simulating a realistic infection process, the evaluation uses publicly available real-world data for both the background and botnet graph.

The following first describes the implementation of the evaluation setup as a modular prototype. Then, the remaining paragraphs detail the respective modules.

Modular Prototype The evaluation utilizes the modular prototype, as illustrated in Figure 6.7, to efficiently simulate botnet infections on large networks. The architecture follows a straightforward design and allows separating the processing of individual steps. This means that the analysis of input data can be performed in a pipelined manner. For example, flows of network communication can continuously be collected in predefined time periods or until a redefined amount of flows is collected. The actual statistical analysis and detection can then happen separately and will not slow down the monitored system while continuing the collection of flows.

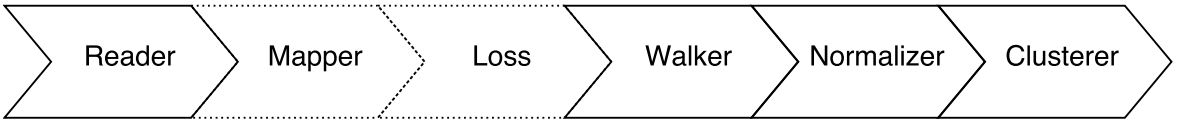


Figure 6.7: The detection prototype is separated into modular services that allow for a distributed data collection and computation. The dotted elements have been added for evaluation purposes.

The main components are as follows. The *Reader* gathers the NetFlow records, parses and converts them into the graph data format. The *Walker* performs n random walks of distance k on the graph's adjacency matrix (cf. Section 4.3.3), which can be computed in a distributed manner since each walk is an independent read-only operation on the graph. The dampening constant is mapped to the resulting probability distribution in the *Normalizer* to reduce the influence of hub

nodes in the network. Finally, the *Clusterer* uses density-based spatial clustering of applications with noise (DBSCAN) to perform the final clustering step.

For the evaluation of the detection algorithm, this prototype is modified. Instead of collecting live NetFlows, the Reader makes use of already captured communication, i.e., G_L. In addition, the prototype is extended by two more steps: (1) a *Mapper* to simulate botnet infections by embedding G_B to creating G_N and (2) a *Loss* step to enforce limited visibility on the network. The modules relevant for the evaluation setup are detailed next in the following paragraphs.

Reader for Data Sources For the evaluation, the *Reader* in the modular prototype utilizes real-world data sets of NetFlows and botnet communication.

- **Carrier Graphs** G_L: Due to the potentially complex structure of the carrier graph, it is not synthesized for evaluation. Instead, two real-world networks from public datasets are used. The first one (labeled *TW07*) is a NetFlow capture from the University of Twente taken in May 2017 [Bar+10], which is converted into a directed communication graph by taking the IP addresses as nodes and each communication flow as a directed edge. The second carrier graph used in the evaluation (labeled *CTU11*) is from a NetFlow capture from the Czech Technical University in August 2011 [Spe+09] and is converted in the same way.

  Although these sets themselves might already include communication of P2P botnets, this is an accepted limitation in the experiments, as this situation is probably the case in the real world.

- **Botnet Graphs** G_B: The data for the botnet graphs is collected from various runs of the StroboCrawler [Haa+16] software on the ZeroAccess and Sality botnets. The crawler is specifically adapted to crawl unstructured botnets as their crawling can be more challenging due to the absence of a structured ID space [Kar+14]. The crawling has been performed on 2016-02-24 for ZeroAccess (*ZA24*), and on 2016-02-25 for Sality (*SA25*). Due to technical restrictions, the Sality botnet dataset only contains superpeers, i.e., only directly reachable nodes. The ZeroAccess dataset contains directly reachable nodes and bots behind NAT.

Mapping for Graph Construction Due to the lack of ground truth in real-world data, infections as simulated as realistic as possible. For that, the *Mapper* in the modular prototype embeds a P2P botnet graph $G_B = (V_B, E_B)$ into a larger carrier graph $G_L = (V_L, E_L)$ (cf. Section 6.4.2.1). The resulting graph $G_N = (V_N, E_N)$ contains G_B as an induced subgraph with $V_B \subseteq V_N$ and $E_B \subseteq E_N$. The graph G_N is assumed a large communication graph based on NetFlow data as potentially collected by an ISP. The algorithm's classification of botnet nodes can afterwards be compared with the ground truth obtained from this mapping process.

The construction of the network communication graph G_N is a transformation $f\colon \{G_L, G_B\} \to G_N$ specified by Algorithm 3. The algorithm simulates the infection process on a copy of G_L – the carrier graph. For each node in the botnet graph G_B, a candidate is uniformly sampled without replacement from G_L, and the relation is saved in the mapping M. After carrying out Algorithm 3, G_N represents the communication among all hosts containing both the legitimate and botnet communication. Note that G_N is still based on a centralized knowledge about all communicating hosts.

Data: G_L, G_B
Result: G_N

initialize G_N as a copy of G_L;
create an empty mapping M;
foreach *v in* V_B **do**
    $u \xleftarrow{R} V_C$;
    add mapping $v \rightarrow u$ to M;
end
foreach *edge* (v_0, v_1) *in* E_B **do**
    $u_0 \coloneqq M[v_0]$;
    $u_1 \coloneqq M[v_1]$;
    if $(u_0, u_1) \notin E_N$ **then**
        $E_N \cup \{(u_0, u_1)\}$;
    end
end

Algorithm 3: The infection mapping.

Loss for Simulating Restrictions For the evaluation of the detection algorithm's robustness against limited vision, it is necessary to remove certain portions of the infected graph. This can be the result of the data collection approach or simply because there is no central NetFlow database for the whole Internet. More details about these restrictions are included in the detection model in Section 4.3.2. The *Loss* module in the modular prototype (cf. Section 6.4.2.1) implements both restrictions by two loss functions taking G_N as input and returning a loss graph G'_N with the respective loss function applied:

1. **Sampling to simulate NetFlow Restrictions**: This restriction focuses on network-based data collection for NetFlow records. The data is usually gathered via the network equipment at various locations, e.g., network backbone, Internet upstream, or peering node. Not all communication between hosts is visible to the network operator. Technically, this loss function is implemented by randomly deleting edges in the graph G_N. The evaluation considers the worst case for the detection algorithm, which is that only botnet communication, e.g., edges $(v_i, v_j) \in E_B$, are removed.

   By applying sampling, the experiments control the portion of randomly chosen edges from E_B labeled as part of a botnet and deleted them from G_N to generate the input G'_N to the detection algorithm.

2. **Host Monitoring to simulate Visibility Restrictions**: This restriction focuses on host-based collection for NetFlow records. On a host basis, all the hosts' communication data is gathered. The visibility depends on the number of hosts that are being monitored. It equals the accumulated vision of the monitored hosts regarding their incoming and outgoing flows. Technically, this loss function is implemented by randomly choosing a subset from the overall hosts in G_N and including all incoming and outgoing edges of the respective hosts to simulate limited knowledge.

   By applying host monitoring, the experiments control n hosts that are randomly chosen from G_N with both their incoming and outgoing edges to generate the input G'_N for the detection algorithm.

Evaluation Data Sets Table 6.11 shows the statistical properties of the utilized carrier and botnet graphs. The botnets have a significantly higher number of edges and a smaller diameter – allowing messages to propagate quickly throughout the network. This means that approaches based on random walks will be effective because the botnet subgraph shows fast-mixing properties. It can be seen that the average botnet node degree is significantly higher than the carrier graph node degrees. Because these connections are preserved throughout the mapping process, the average number of edges added to G_N grows proportionally with the number of edges in the botnet graph $|E_B|$. At the same time, the number of node stays unchanged.

	Carrier Graphs		Botnet Graphs	
	TW07	CTU11	ZA24	SA25
Network Diameter	7	12	5	5
Number of Nodes	66408	38130	4805	1422
Average Node Degree	2.103	2.062	187.415	416.769
Number of Edges	139628	78626	734010	592646
Average Path Length	3.959	2.808	2.163	1.776
Average Clustering Coefficient	7×10^{-4}	3×10^{-3}	0.327	0.605

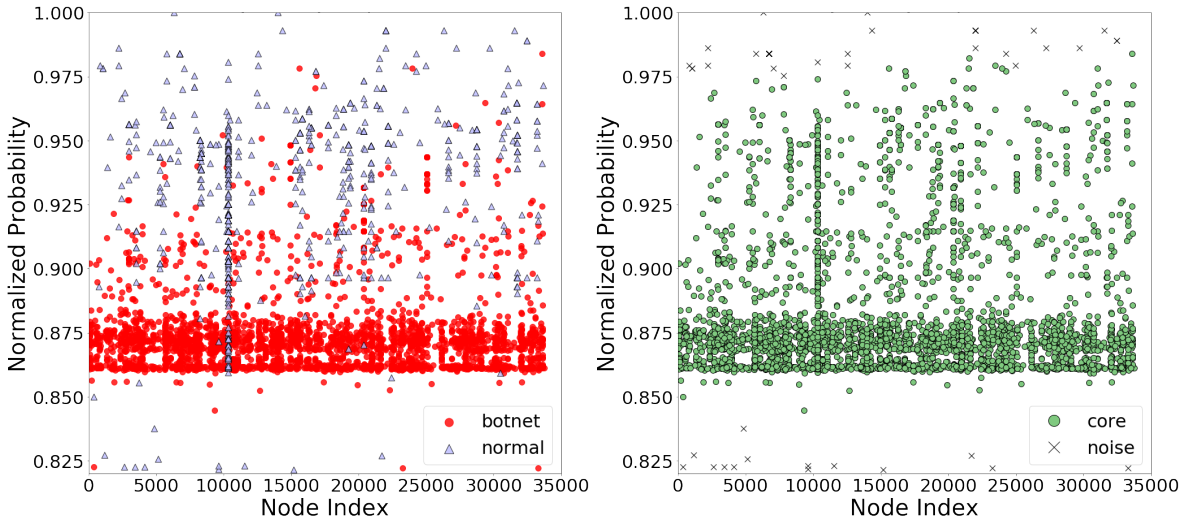
Table 6.11: Statistics on the graphs used during experimentation.

Based on the carrier and botnets graphs in Table 6.11, two combinations are taken for the experiments. For that, *ZA24* and *SA25* are mapped according to Algorithm 3 onto *CTU11* and *TW07*, respectively.

1. **CTU11-ZA24**: The resulting infected graph has 38 130 nodes and 445 267 edges. Figure 6.8 demonstrates how the detection algorithm leverages the fast-mixing property. When performing $n = 10\,000$ random walks of distance $k = 3$, the probability distribution for each node is visualized as a scatter plot in Figure 6.8a. This figure plots each node index against the normalized probability for the respective random walks. Each data point in the scatter plot is categorized based on the ground truth obtained from the mapping process. Botnet participants are shown as red crosses, uninfected nodes as blue dots. Transparency has been applied, so that overlapping data is shown in a more intense color. The normalized botnet data points are clearly separated from the slow-mixing rest.

   Clustering the sanitized data with density-based DBSCAN classifies the largest and densest cluster as a botnet, as illustrated in Figure 6.8b. Core nodes, which are at the center of the classification, appear as green circle markers. It can be seen that the dense area between 0.86 and 0.875 from 6.8a has been clustered. These nodes are considered part of a botnet. The residuals above 0.875 have also been categorized due to imprecise parametrization.

2. **TW07-SA25**: The second infected graph mainly differs regarding the number of nodes in the carrier and the botnet graph. Their proportions differ in an order of magnitude compared to *CTU11-ZA24*. Here, only 2% of the nodes in the resulting infected graph are bots. This lower rate stems from both a bigger carrier network and a smaller botnet graph. This resulting infected graph is meant to test the performance of the detection algorithm on a large-scale network.



(a) The random walk scatter plot with normalized probabilities for the infection of CTU11 with ZA24.

(b) The random walk scatter plot after DBSCAN clustering for the infection of CTU11 with ZA24.

Figure 6.8: Gathered plot examples.

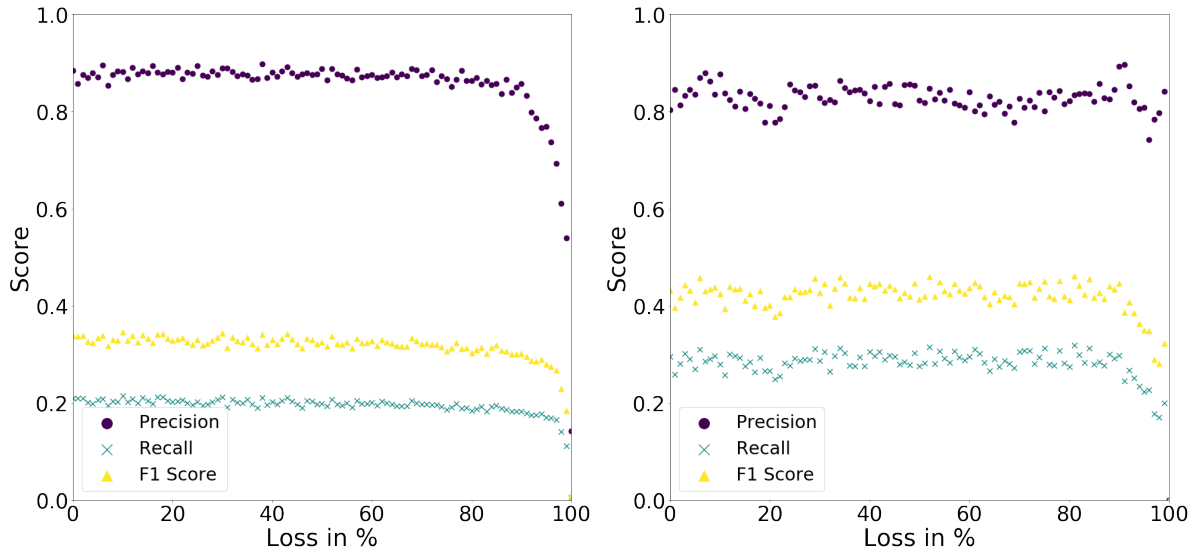
6.4.2.2 Sampling Evaluation

In practice, a network operator has a good view on its own network, but the internal communication relationships between infected machines forming a botnet might not be visible. If these edges do not appear in the communication graph used for analysis, it might negatively affect the precision of the detection algorithm. This effect is investigated by the following experiment that makes use of the loss function *Sampling* (cf. Section 6.4.2.1) for the infected graphs *CTU11-ZA24* and *TW07-SA25*.

To test the behavior of the detection algorithm under different levels of edge loss, 101 runs are performed, each increasing the percentage of removed botnet edges l by one from 0 to 100, according to the *Sampling* strategy. The results are visualized in Figure 6.9, where precision, recall, and the respective F1 score are plotted in dependence on the edge loss parameter. The figure includes result plots for the graphs *CTU11-ZA24* and *TW07-SA25* in Figure 6.9a and 6.9b, respectively.

For Figure 6.9a, it can be observed that even with 90% of botnet edges removed, the algorithm still reaches a precision of more than 83%. Only when over 98% of botnet edges are removed, the precision metric drops below 50%. This result illustrates the robustness of a detection algorithm based on random walks. With the bigger carrier network in Figure 6.9b, it can be seen that the precision varies more. This is because the walk might start at a node that cannot be reached in its set walking distance. Despite this limitation, the reached precision is still well more than 80%, reaching its peak at 90%, even with 91% of the botnet edges being removed.

This experiment shows that even though certain botnet-internal communication relationships are hidden to the operator, it is still possible to extract botnet participants at a significant precision from the larger flow graph.



(a) Scores for graphs CTU11 and ZA24.

(b) Scores for graphs TW07 and SA25.

Figure 6.9: Detection score depending on the random removal of individual NetFlows.

6.4.2.3 Collaboration Evaluation

In most scenarios, the communication of hosts is not limited to a small network, but the communication takes place on an Internet-scale level. The questions are:

- How does the algorithm perform in a single ISP scenario in which the ISP can only analyze its own traffic?
- How many network operators have to collaborate to achieve a satisfying detection performance?

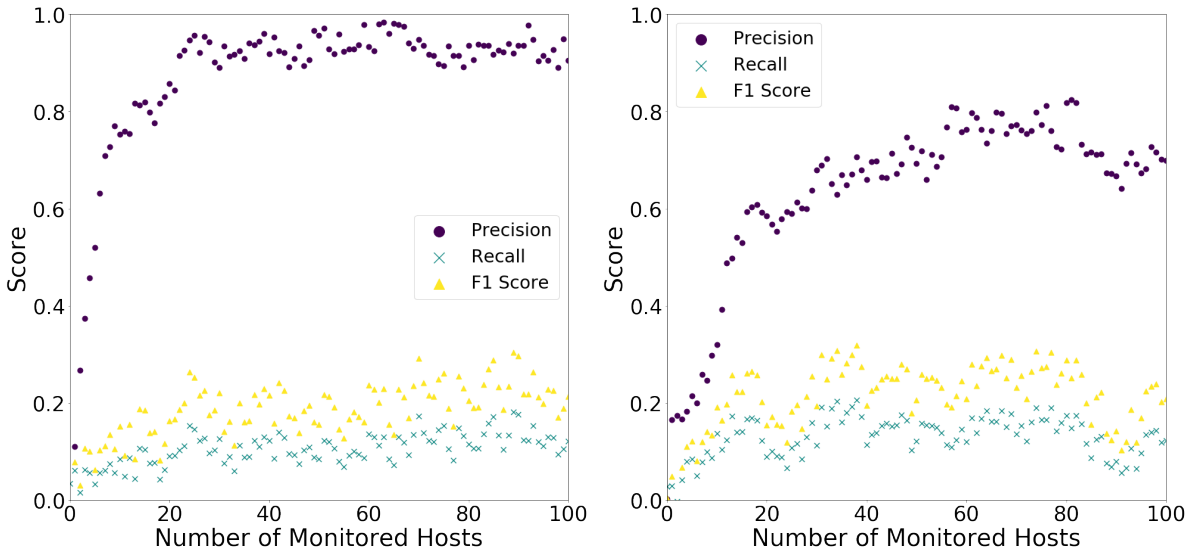
The assumption that a network operator has insight into all the communication of the monitored hosts is modeled by the loss function *Host Monitoring* (cf. Section 6.4.2.1). Thus, this experiment investigates how many monitored hosts n are needed to reach a similar precision as with full visibility in Section 6.4.2.2, which assumes complete knowledge of communication relationships in a botnet.

Based on the same graphs *CTU11-ZA24* and *TW07-SA25*, the loss function for *Network Visibility Restriction* starts with one randomly chosen host to monitor $n = 1$ and is further increased in steps of one until $n = 100$ hosts are monitored at the same time.

The results are visualized in Figure 6.10, where precision, recall, and the respective F1 score are plotted in dependence on the number of monitored hosts. The figure includes result plots for the graphs *CTU11-ZA24* and *TW07-SA25* in Figure 6.10a and 6.10b, respectively.

In general, the number of monitored hosts does not seem to have a significant influence on the recall and F1 scores. The precision in Figure 6.10a, however, increases quite fast to 0.75 with more of monitored hosts until $n = 10$. The precision then continues to increase, but much slower to 0.9 until $n = 25$.

The plots visualize fluctuations in precision, which can have multiple reasons. As the monitored hosts are randomly chosen from G_N, with a too low n, the probability for a monitored host to be either directly connected or in reachable walking distance to a botnet structure is accordingly



(a) Scores for graphs CTU11 and ZA24.

(b) Scores for graphs TW07 and SA25.

Figure 6.10: Detection score depending on the restriction to random monitored hosts.

low. Moreover, with hosts monitored in a large network, they are more likely to be not directly connected and thus not communicating with each other. If the resulting network graph G'_N has a smaller diameter than the random walk distance, it will negatively affect the precision of the algorithm. There are also the trivial cases, where the resulting network graph is a directed path or consists of isolated nodes. However, these cases can be prevented by a strategic selection of sensors or simply increasing the number of monitored hosts.

When comparing Figure 6.10a to Figure 6.10b, it can be seen that, as the carrier network grows from 38 130 to 66 408 hosts, more monitored hosts are needed to provide reliable results. A precision of 0.6 is already achieved for $n = 20$. While a precision greater than 80% can be achieved with 14 monitored hosts for the first carrier network *CTU11*, with the second and bigger carrier network *TW07*, a network of 58 monitored hosts is needed to achieve the same precision.

Although this experiment deals with absolute numbers of monitored hosts, these numbers can be interpreted regarding the sizes of the carrier networks and botnets. Doing so leads to a general rule of thumb how large the set of monitored hosts needs to be to achieve the same precision as with full knowledge on all hosts and their communication. The results of the experiments with real-world data indicate that monitoring less than 1% of the hosts is sufficient as long as this encompasses 5% of the bots.

6.4.3 Summary of Correlated Network Communication

An evaluation of collective detection has been presented for the scenarios scan campaign and P2P botnet. They represent a class of network-wide attack scenarios that target multiple hosts and cause many similar activities, i.e., network traffic. The term collective detection refers to the classification of events. In contrast to classifying events individually, collective detection identifies the attack scenario by the relation among the respective events. The benefits of a collective detection include a solution to two opposed detection problems. The first problem is

that individual malicious events might be detected but their relation and the attack scenario is not identified. The second problem is that individual malicious events might not be detected in the first place and the attack definitively goes unnoticed. How these two problems are solved by collective detection becomes apparent with the two evaluated attack scenarios. The following summarizes their evaluation regarding mitigating restricted visibility to enhance the detection accuracy.

In the first attack scenario, i.e., scan campaigns (cf. Section 6.4.1), scanners with different source IP addresses collaborate in a larger scan campaign. Although each scan probe is eventually detected on its own and all scan probes can easily aggregated per source to scans, the collective detection solves two challenges. First, a scanner with only a few probes could be a false positive or should be filtered because the knowledge gain for the attacker is negligible. The second challenge is to detect coordinated scanners and to reconstruct the scan campaign. Furthermore, having identified a coordinated scanner allows to revise the decision whether it is a false positive or true positive as part of a larger campaign.

The experiments regarding restricted visibility indicate that scan campaigns are still detectable due to similarities among the scan activity of coordinated scanners. The detection accuracy significantly depends on the size of the monitored network as this defines the visibility scope and the scans that are subject to the correlation algorithm. In relation to a backbone network, an ISP captures only 32% of the scanners but still correlates them with an accuracy of 94.5%. For even smaller networks like /24 enterprise networks, the accuracy is significantly affected by the restricted visibility. With only 3.9% of the scanners captured, the accuracy drops to 27.3%.

In the second attack scenario, i.e., P2P botnets (cf. Section 6.4.2), the infected hosts, i.e., bots, establish connections among each other to disseminate commands and updates. Without a signature for the network packets of the specific botnet in question, the classification of individual bot communication fails. The bigger picture of the scenario becomes visible only, when assembling the bots' communication relations to the botnet graph. Based on its characteristics, the entire graph can be classified as malicious, including its bots and their interconnections.

The experiments regarding restricted visibility indicate that, even under worst-case circumstances, P2P botnets are still detectable due to their well-connected nature. For large ISPs, the well-connected botnet structure is still likely to be statistically visible, although they usually sample their NetFlow records by rates 1:128 or 1:512. Anyhow, the experiments indicate that monitoring should encompass at least 5% of the bots. The detection is able to classify bots of unstructured P2P botnets with high precision in large network infrastructures. However, many false negatives might be in the detection result. Nevertheless, the identified bots are true positives with high certainty. Leveraging the communication graph, more potential bots can be revealed and investigated afterwards.

The evaluated scenarios demonstrate that collective detection can withstand or even mitigate the negative effects of restricted visibility on the detection accuracy. Anyhow, collaboration among network sides is motivated, especially for the detection of Internet-wide scenarios. The collaboration extends the network scope and further enhances the detection accuracy.

The previous and this section have evaluated approaches for security monitoring that mitigate restricted monitoring visibility. Their evaluation demonstrated the benefits of correlating events from both heterogeneous and homogeneous data sources to increase the detection accuracy. However, the number of resulting alerts from these and other detection approaches is usually too

large to be analyzed manually. For that reason, the correlation of alerts to a bigger picture is evaluated in the next section.

6.5 Evaluation of Graph-based Alert Correlation

A major challenge of intrusion detection is the handling and analyzing of the IDS alert data. In particular, the large alert volume makes it hard to identify relations among the alerts. Therefore, alert correlation assists in assembling related alerts to a bigger picture of the attack, effectively giving a concise intrusion report. The processing tasks to correlate the alerts are described generically by the alert correlation process in Section 5.1, and a graph-based implementation of the tasks is given in Sections 5.2.1 and 5.3.1. Piecing these two implementations together along the alert correlation process, results in the *graph-based alert correlation* (GAC) approach.

As it is with most approaches for alert correlation, also GAC is based on feature similarity among the alerts. However, clustering the alerts is only the first step in GAC and results in a reduction of the alert volume. For a concise intrusion report, GAC further links the alert clusters to multi-step attacks. What is particularly special in GAC is that it leverages additional context about the alert clusters for the linking.

This evaluation of GAC investigates how community clustering in particular helps to identify distributed attacks in large alert sets. Furthermore, this evaluation investigates the scenario definition in GAC as context for the reconstruction of multi-step attacks, especially under the assumption of false positive alerts.

This section revises parts of the conference paper [HF18]. The section first evaluates every stage of GAC according to the alert correlation process separately for correct working of the correlation algorithm. The second part of the evaluation then demonstrate the result of GAC when operating on real-world data without ground truth.

6.5.1 Stepwise Analysis of Process Stages

This first evaluation part creates artificial data to have both the control and the ground truth about the data that is used for the analysis of GAC. Only by doing so, analyzes the behavior of every correlation step systematically. For that, this stepwise analysis of GAC first describes the data set as input for the experiments. Then, the performance of alert clustering and attack scenario identification of GAC are conducted in independent experiments for analyzing their correctness.

6.5.1.1 Artificial Alert Data for Analysis

Although being aware of criticism of using artificial data for testing intrusion detection [McH00], note that the scope of GAC is not intrusion detection but alert correlation. Moreover, it is not about generating a data set representative for the real world, but about generating input to analyze the effectiveness of what the algorithm was designed for.

Input for the analysis of GAC consists out of alerts from one or more artificial attacks. Depending on the process step that is analyzed, the alert data is modified to generate input variations for

the experiment. As GAC is designed to detect distributed attacks, the attack generation makes uses of the attack patterns according to Table 6.12. This table defines which IPs and ports are equal for all alerts of the same attack and which are probably different and can be determined randomly. Since GAC requires no knowledge about subnets and type of port, generating alerts uses the full IP (0.0.0.0 to 255.255.255.255) and port range (0 to 65535). Based on these features and pattern, different kinds of attacks are generated:

- In a horizontal *Scan* (`OtM`), one source targets several hosts on the same destination port.
- In a *distributed denial-of-service (DDoS)* attack (`MtO`), several sources target the same IP and port combination from arbitrary source ports.
- In a *Worm* (`MtM`) spreading, each host targets a subset of other hosts on a fixed destination port.

Scenario	SrcIP	SrcPort	DstIP	DstPort	Example	Similarity
`OtO`	X		X		Vertical Scan (V-Scan)	2/4
`OtM`	X			X	Horizontal Scan (H-Scan)	2/4
`MtO`			X	X	Service DDoS	2/4
			X		Distributed V-Scan	1/4
`MtM`				X	Worm	1/4

Table 6.12: Similarity values among alerts of different attack scenarios.

In reality, it is assumed that several attacks happen at the same time. Thus, the experiments generate alerts for several attacks independently, resulting in alerts for particular attack instances. Then, they are partially merge by mapping two hosts from different attack instances to the same host, i.e., replacing IP X from the first and IP Y from the second alert set by a third IP Z. The *overlapping parameter* determines the fraction of IPs that overlap. This input impedes alert clustering to isolate alerts into groups of different attacks.

In addition, the experiments blur alert data via a *blurring parameter* β to create additional alerts that represent isolated single alerts or false positives, e.g., as a result of an inaccurate sensor. Blurring adds no alerts if $\beta = 0$ and adds more random alerts until $\beta = 1$. Blurring not only adds additional alerts for the n existing hosts in the alert data, but also creates alerts for new hosts. This is achieved by the way that blurring is implemented. It starts with randomly sampling the existing attacking and target hosts. The set of $\beta \cdot n$ sampled attackers and $\beta \cdot n$ sampled targets is each extended by another $\beta \cdot n$ new hosts. Pairwise for the next attacker and target from the extended sets, an alert is added until an alert is generated for every host in both sets. As a result, on average $\beta \cdot n$ new hosts and $1.23 \cdot \beta \cdot n$ new alerts are added.

While the experiments for clustering analysis merge the alerts for multiple attacks (via overlapping parameter), the experiments for scenario identification create separate data sets. Each of them contains a single attack plus additional alerts as false positives (via blurring parameter) in form of a cluster to simulate an inaccurate clustering or an inaccurate sensor. The specific parameterization to generate the alert data used in the evaluation is described in more detail together with the results.

6.5.1.2 Alert Clustering in Alert Similarity Graph

The first analysis experiments are regarding the alert similarity graph G_{attr} (cf. Section 5.2.1) and the clustering of alerts to attack steps. The minimum similarity threshold τ is used to filter edges in G_{attr}. The threshold is set to $\tau = 0.25$ as this is the minimal value at which related alerts stay connected for attacks in Table 6.12. On the filtered graph, community clustering is applied to potentially result in one cluster per attack (step). The question is how to set the clique size parameter k that determines the minimum community size. In particular, the analysis results should indicate how to set k to achieve best clustering performance. Technically, the clustering method clique percolation method (CPM) requires clique size $k \geq 2$. And obviously, it should be $k \leq |G_{\text{attr}}|$, i.e., not larger than the number alerts in the graph, to allow mining of any cluster. More specific, k should not be larger than the number alerts from the smallest attack, i.e., $min(|S_i| \text{ for } S_i \in \hat{S})$. The following experiments investigate the performance between the lower and upper bound of the range $k \in [2, |G_{\text{attr}}|]$.

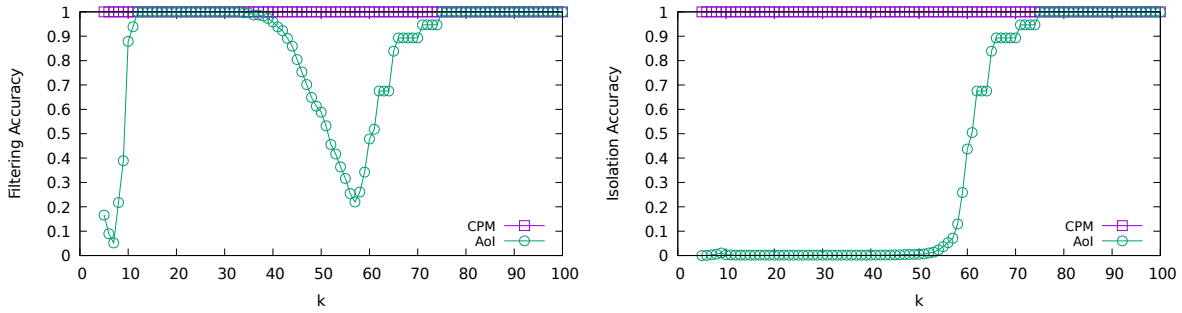
Apart from showing how graph-based clustering performs, the CPM method in GAC is compared to a similar approach that is denoted as Attribute-oriented Induction (AOI) [Jul03] (cf. root cause analysis in Section 3.4.2). AOI is an efficient algorithm to find attribute patterns in a set of alerts to establish clusters of minimum size k, similar to GAC that employs community clustering and forms cliques of at least size k. The following experiments report the average results across 16 runs.

Metrics For *alert filtering* and *attack isolation* (cf. alert clustering in Section 5.1), no metrics have been found that reflect the clustering performance to both challenges appropriately. Therefore, this experiment applied two custom methods for evaluating both challenges individually. They are incorporated into the standard metric accuracy $ACC = \frac{TP+TN}{P+N}$ that compares the ground truth to one identified cluster.

Alert filtering accuracy should only consider the union of alerts in original attacks as ground truth $GT = \bigcup_{S_i \in \hat{S}} \bigcup_{a_j \in S_i} a_j$ and the union of alerts in all identified clusters as detected attacks $DA = \bigcup_{C_i \in \hat{C}} \bigcup_{a_j \in C_i} a_j$. The accuracy is used to evaluate the performance of DA modeling GT by comparing these two sets of alerts.

Attack isolation accuracy should evaluate the performance of $\hat{C}$ modeling $\hat{S}$. It describes a correct grouping with respect to all original attacks and clusters of alerts. As the accuracy ACC can only compare one attack and one cluster, the comparison is broken down to individual pairs of $\hat{C}$ and $\hat{S}$. This method aims to find which alert cluster $C_j \in \hat{C}$ models which original attack $\S_i \in \hat{S}$ best. The accuracy ACC is calculated for every possible combination. Hence, a mapping between $C_i \in \hat{C}$ and $S_j \in \hat{S}$ is derived under the constraint that each attack and cluster can only be mapped once. A greedy algorithm creates a mapping such that either only single attacks or single clusters are left unmapped. An accuracy of 0 is defined for any unmapped cluster or attack, respectively. The total accuracy is then the average accuracy for all mapped and unmapped attacks or clusters, respectively.

Clustering Alerts from Single Attacks Figure 6.11 illustrates the clustering results for alerts from an unmodified DDoS attack, in which 10 hosts attack one target. The alert set consists out of 500 alerts, so 50 alerts are generated per source on average. Therefore, the accuracy for alert filtering (Figure 6.11a) and the attack isolation (Figure 6.11b) drops to zero for $k > 500$



(a) Accuracy for alert filtering.

(b) Accuracy for attack isolation.

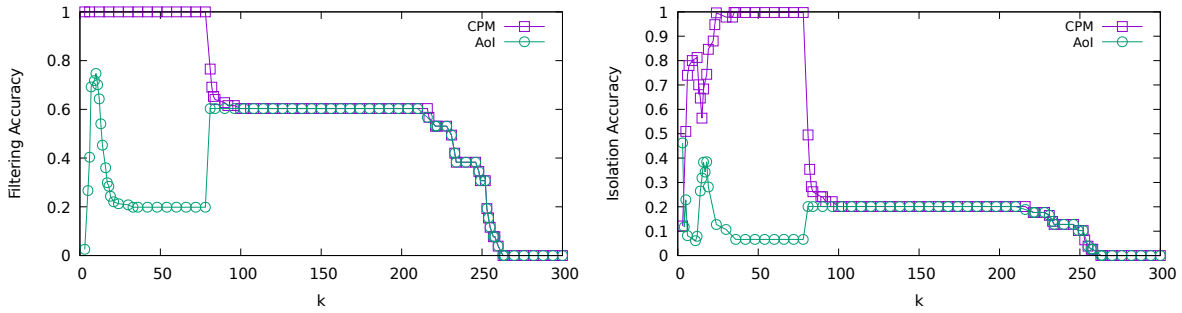
Figure 6.11: Accuracy for clustering alerts from a DDoS attack depending on clustering parameter k.

for both the community clustering (CPM) in GAC and AOI. For lower values, AOI tries to mine clusters while generalizing as few attributes as possible. If an AOI cluster, i.e., attribute pattern, itself consists of many smaller patterns, AOI will report these smaller clusters if k allows. This behavior is a problem when using AOI on alerts from distributed attacks, e.g., when each source in a DDoS attack causes multiple alerts. This is the reason why the attack isolation accuracy of AOI is below 0.3 for $k \leq 60$ and why the alert filtering accuracy of AOI drops for $60 \leq k \leq 75$. Community clustering, instead, can combine these smaller clusters, i.e., k-cliques, to communities and therefore achieves an accuracy of 100% for both filtering and isolation.

The advantage of community clustering, therefore, is that it potentially works even though k is much lower than the smallest attack. However, k has to be high enough to filter noise and to split alerts of loosely coupled attacks. This problem is investigated in the next experiment.

Clustering Alerts from Overlapping Attacks The clustering results for three attacks overlapping 30% of their IP addresses are plotted in Figure 6.12. There is one DDoS with 80 alerts, one Scan with 80 alerts and one worm for 20 hosts with a spread factor of 0.7 (266 alerts on average). Other experiments showed a linear dependency of the clustering parameter k on the presence of false positive alerts. The accuracy of alert filtering (Figure 6.12a) and attack isolation (Figure 6.12b) are plotted depending on the parameter k. Community clustering (CPM) in GAC has an alert filtering accuracy of 100% for $3 \leq k \leq 80$, i.e., for any k that is less or equal than the smallest attack size. The range of k to achieve an accuracy of attack isolation of (almost) 100% is different for the lower bound and requires $24 \leq k \leq 80$. Values for $k \leq 24$ make the CPM to merge alerts of different attack steps into one cluster because of the overlapping IP addresses. CPM achieves at least equal accuracy like AOI in the first place and can even cluster all attacks correctly. This is because CPM achieves 100% accuracy for alert filtering and attack isolation for the same values of k. AOI instead has its maximum accuracy of 73% for alert filtering at $k = 8$ and 38% for attack isolation at $k = 18$ Thus, choosing a value for k in AOI is always a balance between good alert filtering or attack isolation.

The experiment for clustering alerts of overlapping distributed attacks indicates that (1) GAC achieves higher accuracy than AOI and (2) the clustering parameter k allows to achieve these high accuracy for a wide range of the parameter.



(a) Accuracy for alert filtering.

(b) Accuracy for attack isolation.

Figure 6.12: Accuracy for clustering alerts from overlapping DDoS, scan, and worm attacks depending on clustering parameter k.

6.5.1.3 Scenario Identification on Alert Flow Graphs

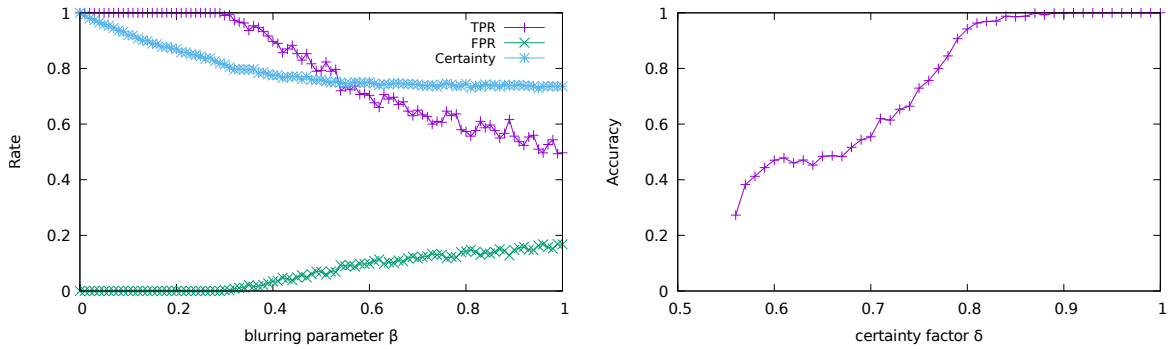
GAC classifies attacks represented in the form of clustered alerts as `OtO`, `OtM`, `MtO`, or `MtM` (cf. Section 5.3.1.1). Such an attack characterization must be error tolerant, because there might be false positives or false negatives when clustering alerts. For that, the scenario metrics $\delta_{\texttt{OtO}}, \delta_{\texttt{OtM}}, \delta_{\texttt{MtO}}, \delta_{\texttt{MtM}}$ are designed to represent the confidence in labeling an alert cluster with the respective scenarios. The metric with maximum value determines the scenario.

The following experiments evaluate the robustness of attack scenario identification on the basis of artificially created attacks on which blurring (cf. Section 6.5.1.1) is applied to simulate false positives in clustering. The data set contains instances of a DDoS attack with 300 attackers, a scan attack with 300 targets, and a worm attack with 75 hosts and spreading factor 0.7. For one repetition of an experiment, the three resultant attack clusters are each blurred with a specific parameter β. The three blurred attack variants are then classified by GAC and they are expected to be identified as `MtO`, `OtM`, and `MtM` respectively.

Effect of False Positive Alerts on the classification The first experiment investigates the performance of scenario identification in presence of inaccurate clustering. To simulate this, the blurring parameter β varies in between zero and one in steps of 0.01. The complete experiment is executed 100 times, such that for every value of β, 100 blurred variants of the three attacks are generated. This is necessary because blurring introduces randomness when generating false positive alerts for a specific value of the blurring parameter.

The true positive rate (TPR) and the false positive rate (FPR) depending on β are plotted in Figure 6.13a as well as the average certainty δ. The figure shows that scenario identification in GAC can completely tolerate false positive alerts, i.e., still achieve 100% accuracy, to some extent. It also shows that it can be TPR $= 1$ and the FPR $= 0$, even though the certainty $\delta < 1$. The TPR starts to decrease once the blurring parameter exceeds $\beta \geq 0.3$.

The experiment results also indicate that classifying DDoS and scan attacks is more sensitive to blurring than classifying worm attacks. When $\delta = 1$, the metric $\delta_{\texttt{MtO}}$ for a DDoS attack is 0.52, while the metric $\delta_{\texttt{MtM}}$ for a worm attack is 0.79. As a consequence of the bias towards $\delta_{\texttt{MtM}}$, blurred worm attacks are always successfully identified as `MtM`. On overage, the certainty δ in identifying the three attacks over the 100 repetitions drops to 0.73 in Figure 6.13a. The TPR drops to 0.50 and the FPR stays below 0.17.



(a) Detection rate depending on blurring.

(b) Accuracy depending on certainty.

Figure 6.13: Accuracy for labeling alert clusters with the attack scenario.

Interpretation of the Classification Certainty The second experiment asks about the expected accuracy when a scenario is identified with a specific certainty δ. For this reason, the three attacks are again blurred with a random $\beta \in [0,1]$, which is repeated with a different β 10000 times. The average accuracy, i.e., fraction of correct classifications, over all repetitions was 80%.

To detail on the accuracy, Figure 6.13b plots the average accuracy depending on the certainty δ. During the repetitions, the scenario classifications were grouped in bins of 0.01 regarding their resultant certainty δ and the average accuracy is calculated per bin. The actual observed certainty values range from 0.56 to 1. As expected, a high certainty leads to high accuracy. More specific, 100% accuracy is always achieved when the certainty $\delta \geq 0.91$, which is the case for 20% of the attack clusters. 53% of the clusters have been classified with a certainty $\delta \geq 0.79$. The average accuracy for classifications with this certainty was 90%.

The experiments show that the majority of attacks can be correctly classified even in worst-case situations. Also, false positive scenario labels are expected only in classifications with high uncertainty.

6.5.2 Real-World Evaluation

The second part of the GAC evaluation performances on real-world data. For that, the full algorithm is applied to the real-world data set that is introduced first. Afterwards, the results of clustering, labeling, and interconnection to multi-step attacks is reported and discussed.

6.5.2.1 Real-World Alert Data Set

The SANS Internet Storm Center operates a community-based collaborative monitoring system for Internet threats on a global level. This system is called DShield[3] and collects network incident logs, i.e., alerts, from various contributors. Every alert in DShield contains the IP and port of the source and the target, among others. However, the target IP is hashed for anonymity reasons. This real-world evaluation correlates DShield alerts from the days given in Table 6.13. The table summarizes each set by the number of alerts per day, from both the years 2016 and

3. https://www.dshield.org/

2017 to enable a representative view, and additionally reports the number of distinct attacking and targeted IP addresses.

GAC performs on a finite set of alerts, not on an alert stream. Therefore, the data sets of 24 hours is split into alert batches, each being a separate input to an individual processing with GAC. Experiences from other runs have shown that batches of 5000 subsequent alerts are a reasonable size as complexity increases heavily with graph size. Anyhow, processing the alerts in batches enables parallel runs. For that, the experiment use 16 workers in parallel on a system with 2×8 2.2 GHz cores and 128 GB RAM. With this setup, it takes on average 1 to 1.5 minutes per chunk depending on the data set.

Set	Day	Alerts	Sources	Targets
1	2016-08-22	4517498	395872	100675
2	2016-08-23	7238861	443981	116965
3	2016-08-24	5825579	427718	116835
4	2016-08-25	5125589	406530	100053
5	2017-08-11	3668198	281690	17531
6	2017-08-12	3937357	345382	17497
7	2017-08-13	4138175	329108	17155

Table 6.13: Days of DShield sets and their statistics.

As in Section 6.5.1, the parameter for the similarity threshold in GAC is set to $\tau = 0.25$ to require at least one of four attributes to be equal between two alerts. CPM is based on the clique size $k = 15$ such that the smallest attack consists out of 15 similar alerts to filter false positive alerts. The results in the following experiments represent the aggregation over all batches of the specific day.

Although the following describes the outcome of the different stages, note that the results from the real-world evaluation are regarding a full run of GAC, starting with IDS alerts from the DShield data set as input and ending with the presentation of multi-step attacks in the attack graph.

6.5.2.2 Clustering

The outcome of clustering has two properties that are important to evaluate. First, the alerts that are not clustered and second, the grouping of the remaining into clusters. Table 6.14 shows a classification for the assignment of alerts and the fraction of nodes per class. An alert with assignment *None* was filtered and therefore not considered further. The assignment *Multi* is specific for the CPM clustering and describes alerts that are grouped into more than one cluster. Furthermore, Table 6.14 shows the number of clusters as well as the average size and standard deviation among the clusters.

The low fraction of unassigned alerts shows that GAC encompasses almost all alerts in its results. This is expected because the nature of distributed attacks causes a lot of alerts, whereas a targeted attack like APT causes only few alerts which are distributed along a large time frame. The large amount of clusters and the high standard deviation is because of the many different clusters and shows that GAC is not limited to a specific form of clusters.

	Assignment			Cluster Size	
Set	**None**	**Multi**	**Clusters**	**Avg**	**Std**
1	0.70%	9.33%	25111	195.83	776.88
2	0.84%	6.42%	31242	244.94	871.69
3	0.90%	7.33%	27235	228.02	836.50
4	0.71%	9.70%	29654	188.71	754.80
5	1.56%	21.22%	44664	98.62	459.29
6	1.37%	21.72%	49188	96.67	452.65
7	1.38%	19.67%	47289	103.78	477.97

Table 6.14: Clustering statistics for DShield sets.

6.5.2.3 Attack Scenario Identification

At this stage of GAC, each cluster is analyzed to detect the attack scenario. Table 6.15 describes the median value that was used as certainty δ, its average value, and standard deviation. The table also shows the fraction of clusters that are identified as `OtO`, `OtM`, and `MtO`.

	Certainty Factor δ			Scenario		
Set	**Med**	**Avg**	**Std**	**OtO**	**OtM**	**MtO**
1	1.0	0.918	0.143	12.50%	20.75%	66.75%
2	1.0	0.911	0.147	14.53%	24.79%	60.68%
3	1.0	0.909	0.147	14.65%	22.63%	62.73%
4	1.0	0.923	0.140	11.37%	21.18%	67.45%
5	1.0	0.922	0.142	13.84%	18.37%	67.79%
6	1.0	0.930	0.137	12.32%	16.89%	70.79%
7	1.0	0.927	0.140	13.16%	17.56%	69.28%

Table 6.15: Scenario identification statistics for DShield sets.

The certainty factor δ in Table 6.15 shows that the majority of values is expected to be between 1 and 0.75. This allows a high confidence in the identification, as Section 6.5.1.3 indicated this range to achieve good results in theory. This is also manually verified with a sample of labels.

Most of the time, clusters whose certainty in scenario identification is $\delta < 0.75$ are caused by alerts with same destination port. The *alert flow graph* then usually consists out of several unconnected components, where groups of IPs appear to represent unrelated attacks without any connection among them apart from the same destination port. When these clusters consist out of small components with a few hosts each, they are tagged with `OtO`. If they additionally include a large component of many hosts, this component usually follows the properties of `OtM` or `MtO` and the cluster is tagged respectively.

No worm-like attacks, i.e., `MtM`, were observed during the experiment. Independent from the existence of such an attack in the data set, it is not possible to detect it in the DShield data set. This is because the hashed target IPs in DShield prevent to have consistent identifiers across attacking and targeted hosts. However, GAC was able to label every cluster, and the majority of scenarios ($> 85\%$) were identified as `OtM` or `MtO`.

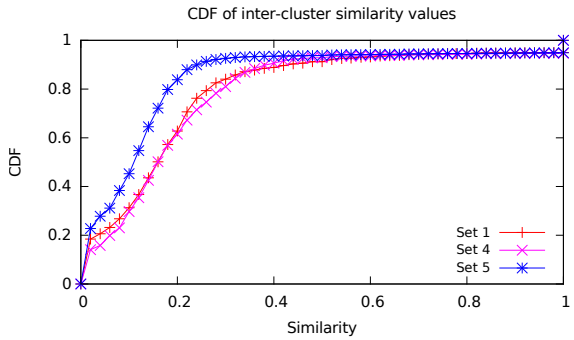


Figure 6.14: CDF of similarity values during the multi-step detection in the DShield sets.

6.5.2.4 Multi-Step Detection

The last stage of GAC assembles multi-stage attacks, i.e., calculates the similarity of IP addresses among the clusters to reveal relations between attacks. The result of this multi-step detection is illustrated in Figure 6.14 in form of a cumulative distribution function (CDF) plot. It shows the value distribution of the inter-cluster similarity. All sets show similar results, however, only the day 1, 4, and 5 are included in the figure for demonstration purposes.

The inter-cluster similarity in the plot is in almost all cases either 1 or < 0.5. Any two clusters rarely have a similarity ≥ 0.5 and < 1. So one can say that if two clusters have a high similarity, i.e., above 0.5, it is most probably 1. Despite the low value of about 5% high inter-cluster similarities, it shows that GAC can detect significant overlap in attacks with probably high precision. This is especially useful as it eases human analysis by clearly pointing out these multi-stage attacks. In the DShield data sets, high similarities are usually caused by clusters that have both high similarity among attackers (sim_{AA}) and victims (sim_{VV}). Anyway, note that potentially not all multi-step attacks could be detected because the hashed target IP addresses in the data sets prevent to identify hosts that are both attacked and themselves attackers.

6.5.3 Summary of Graph-based Alert Correlation

An evaluation of GAC has been presented for the correlation of alerts from distributed and multi-step attacks to concisely summarize the network-wide intrusion. Without the assembled view of the alerts for representing the whole attack, the attack is probably underestimated and cannot be mitigated effectively.

In the first evaluation part, GAC has been analyzed stepwise with specially crafted input for the different stages in the alert correlation process to test their correct behavior. For the detection of distributed attacks through alert clustering, the analysis results indicate that the clustering parameter τ can be chosen from a much broader range compared to other state-of-the-art alert clustering approaches. In particular, GAC shines at both filtering alerts that are not related to any larger attack and isolating alerts from different attacks. This then exactly results in clusters that contain only alerts that are all related to each other. Furthermore, the labeling of alert clusters with their attack scenarios for the linking of attack steps has been tested. The simulation of false positive alerts in clusters highlights the role of the classification certainty δ to express the trust in the identification of a particular scenario. Any classification with more then 91% certainty should be safe to be accepted, and even a certainty of 79% still yields very good results. Because

of both the accurate alert clustering and scenario labeling, the interconnection of attack steps to multi-step attacks seems very promising.

The outcome of the full multi-step detection by GAC on real-world data has been analyzed in the second evaluation part. The alert clustering is able to assign about 99% of the IDS alerts to a cluster. The remaining 1% are definitively not part of a larger attack. Apart from those clusters that clearly follow the alert pattern of a distributed attack, some other clusters exists that seem to have some chaotic collection of alerts. Even though these clusters do not represent a distributed attack, they are likely to contain many false positives and unrelated alerts that can be manually filtered all together at once. Such chaotic alert clusters might also be filtered based on the scenario label or, more specifically, on the labeling certainty. Labeling the real-world alert clusters achieves 92% certainty on average, which is within the range of correct classifications according to the analysis results of the first evaluation part. The most prominent scenario is Many-to-One (`MtO`). The results of the multi-step detection in the last step indicate that GAC effectively leverages the scenario labels to precisely link attacks with overlapping IP addresses.

In summary, GAC enables alert correlation for the detection of distributed and multi-step attacks that potentially threaten the whole network. The evaluation has successfully tested GAC regarding both its correct working and its applicability on real-world data. However, note that GAC is supposed to cluster alerts from attacks with many alerts only, i.e., bulk attacks. Alerts from more stealthy attacks are expected to be filtered. To still enable their detection, the next section evaluates the continuous aggregation of filtered alerts.

6.6 Evaluation of Weak Alert Correlation

IDS alerts of APT and other stealthy attacks usually occur infrequently over a long time period. Regular alert clustering probably fails to identify the relations among these weak alerts. For that, *weak alert correlation* (cf. Section 5.2.2) works on top of other alert clustering approaches that identify bulk attacks. In an online fashion, weak alert correlation consumes the stream of unclustered alerts to aggregate them to APT-like attack steps.

This evaluation of the weak alert correlation investigates how the continuous aggregation of weak alerts helps to identify slow and stealthy APT attacks. In particular, this evaluation analyzes to which extent the APT characteristics network direction, alert frequency, and attack topology can be incorporated and leveraged in the detection of APT attack steps.

This section revises parts of the supervised master thesis [Ort19]. The section first constructs an APT scenario from real-world traffic data that is the running example to be detected throughout this evaluation. In fact, this section applies the weak alert correlation in two different ways to demonstrate its detection capabilities. For that, the second section evaluates the algorithm on weak alerts that remain after regular alert clustering. The third section, alternatively, evaluates the algorithm on the full and unfiltered alert set.

6.6.1 Scenario Construction

The evaluation of the weak alert correlation constructs an APT scenario from real-world attacks and embeds this into a background data set with benign and other malicious traffic.

APT Scenario The constructed APT scenario in this evaluation takes place in a small network that is made up of a single network zone with 10 nodes, denoted as zone `Intranet` with an IP subnet of 172.16.0.0/24. Figure 6.15 illustrates the individual steps of an APT attack performed over seven days. It starts with the `Internet` host 10.99.99.8 that uses a remote code execution vulnerability based on *EternalRomance*. Afterwards, 172.16.0.1 is controlled by the attacker and requests another piece of malware from 10.99.99.189. The malware persists. Two days later, the infected node engages in command and control (C2) with 199.231.188.109. Another two days later, 172.16.0.1 uses *PS-EXEC* – a tool for legitimate Windows remote administration – to move laterally to 172.16.0.5 via the Server Message Block (SMB) protocol. Finally, on the last day of observation, the newly infected node 172.16.0.5 attacks a third internal node (172.16.0.10) with an SQL injection (and web injection) attack.

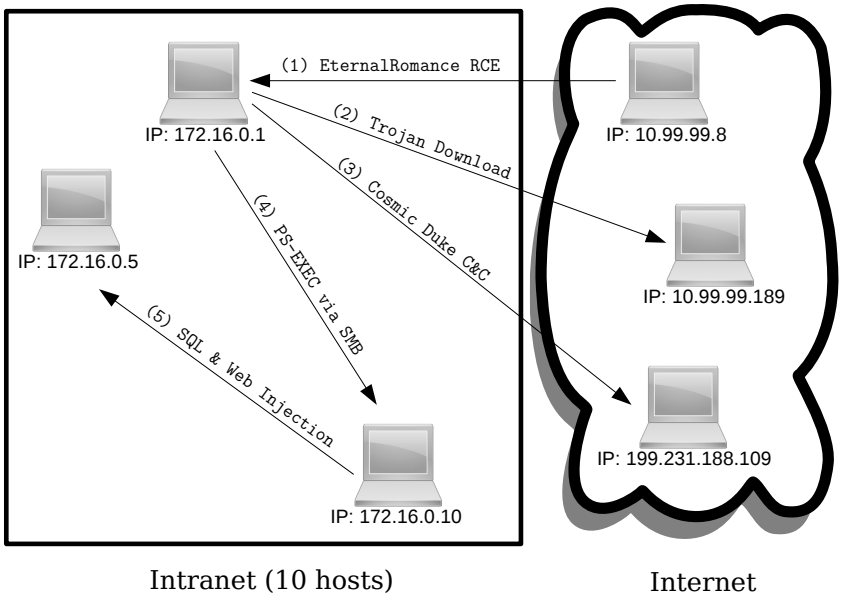


Figure 6.15: Attack steps of the constructed APT attack.

Mixing Data Traffic The traffic data set is based on the *CSE-CIC-IDS2018* data set of the University of New Brunswick[4]. Hosts in their data set run different operating systems and establish connections to the `Internet` using well-known applications and protocols. This set contains full packet payloads and also already includes some noisy attacks such as brute-force password guessing or (D)DoS. Out of their set, the incoming and outgoing traffic of 40 monitored IP addresses is extracted. In addition, some of these systems are consolidated by rewriting IP addresses such that the resulting data set ended up with an `Intranet` of 10 communicating machines. This data set is used for background traffic and noise.

On top of this traffic base, specific attack steps for the reflection of the APT scenario is embedded. The PCAP data for the attack steps have been taken from the *CSE-CIC-IDS2018* data set of the University of New Brunswick (Web Injection), from *ericconrad.com*[5] (shadow broker's *EternalRomance* and *Trojan Download*), the *Data Exfiltration Malware* samples from the

4. https://www.unb.ca/cic/datasets/ids-2018.html
5. https://www.ericconrad.com/2017/04/shadowbrokers-pcaps-etc.html

University of Twente[6] (Cosmic Duke C2), and from *github.com/401trg*[7] (PS-EXEC). The final data set, i.e., the mix of background traffic and the APT attack steps, results in a total of 7.7 GB of PCAP data.

Generating IDS Alerts The Zeek IDS was used to analyze the final PCAP data. The analysis with Zeek lead to 250,521 IDS alerts. Table 6.16 shows those IDS alerts that belong to the known attacks – the ground truth in this experiment. As Zeek has no detection scripts for the specific attacks in the ground truth, this evaluation enables some of them to be detected by custom scripts. However, neither the *EternalRomance* exploit nor the C2 traffic triggered any alerts in Zeek. Nevertheless, as the alert prefix indicates, relevant alerts have been generated by the custom scripts.

Attack	IDS Alert Type	# Alerts
EternalRomance RCE		0
Trojan Download	Conn::Content_Gap	16
	Custom::Windows_Executable_Download	1
Cosmic Duke C2		0
PS-EXEC via SMB	Custom::SMB_Executable_File_Transfer	1
SQL & Web injection	Custom::Javascript_Web_Injection_URI	73
	Custom::SQL_Web_Injection_URI	20
	Custom::Web_Login_Guessing	7
Total		118

Table 6.16: Ground truth in the 250,521 Zeek IDS alerts.

Furthermore, DECANTeR analyzed HTTP sessions in the PCAP data specifically for the detection of C2 traffic. Based on the *http.log* file produced by Zeek, the first 10,000 lines are used for training and the remaining 97,013 lines are used to detect 34 HTTP anomalies in total. One of them in fact describes the malicious C2 traffic as part of the APT attack.

Generating Meta Alerts and Weak Alerts Traditional alert correlation is used to correlate the alert corpus and to filter *weak* alerts. For that, GAC has been leveraged as correlation function $\mathscr{K}$ and parameterized it with a batch size of 5000 and a minimum cluster size of $k = 15$ (cf. Section 6.5). It produced a total of 370 meta alerts. The correlation yields interesting results. GAC itself was already able to correlate one of the attack steps correctly. It clustered those 17 IDS alerts that belong to the `trojan download` into one single meta alert. However, GAC also made mistakes. It produced two meta alerts, which both contain some of the alerts that belong to the `web injection` attack. Moreover, these two meta alerts even have different alert types. One meta alert is correctly clustered as type *one-to-one*, the other meta alert, however, has the type *one-to-many* and groups two different victims. Hence, that meta alert groups true positive IDS alerts together with false positives. Lastly, GAC clustered some IDS alerts twice as part of different meta alerts. Table 6.17 summarizes the results of the GAC correlation. In addition, it includes the C2 pattern alert reported by DECANTeR.

6. https://www.utwente.nl/en/eemcs/scs/downloads/20171127_DEM/
7. https://github.com/401trg/detections/tree/master/pcaps

Attack	Number of			Algorithm
	Attackers	Victims	Alerts	
EternalRomance RCE			0	
Trojan Download	1	1	17	GAC
Cosmic Duke C2	1	1	1	DECANTeR
PS-EXEC via SMB			0	
SQL & Web injection	1	1	49	GAC
	1	2	47	GAC
Total			114	

Table 6.17: Ground truth in the 404 IDS meta alerts

The correlation with GAC left 1,593 IDS alerts uncorrelated that are consequently marked as weak. Ten of them are related to the ground truth attacks. In particular, one alert of type *Custom::SMB_Executable_File_Transfer* is related to the `PS-EXEC via SMB` attack. The remaining nine alerts belong to the `SQL & Web injection`, that consist out of seven *Custom::Web_Login_Guessing* and two *Custom::Javascript_Web_Injection_URI* alerts.

6.6.2 Weak Alerts Evaluation

The remaining 1,593 alerts are now subject to correlation of weak alerts. Those alerts were filtered by GAC clustering, transferred to host- & zone-communication graph (HZCG), extracted as neighbor groups, and continuously accumulated to aggregations across 51 alert batches. Resulting aggregations exist for three different directions:

- `Internet` → `Intranet` (incoming attacks)
- `Intranet` → `Internet` (outgoing attacks)
- `Intranet` → `Intranet` (internal attacks)

The aggregations for the third direction, i.e., for internal attacks, contain exactly the ten alerts that belong to the `SQL & Web injection` and the `PS-EXEC via SMB` attack. In fact, two aggregation objects exist, one for each attack. The results of generating weak meta alerts for this direction leads to accurate results as illustrated in Table 6.18.

Attack	*Label*	*Attackers*	*Victims*	# Alerts
PS-EXEC via SMB	`OtO`	172.16.0.1	172.16.0.5	1
SQL & Web injection	`OtO`	172.16.0.5	172.16.0.10	9

Table 6.18: Weak meta alerts for direction Intranet → Intranet.

Of more interest for clustering aggregations of weak alerts with DBSCAN are the other two directions (`Internet` → `Intranet` and `Intranet` → `Internet`), as they together count 1,583 weak alerts. However, they include no alerts of the ground truth attacks, so their accuracy cannot be evaluated here. Instead, this experiment reports the final outcome of generating weak meta alerts for all three directions.

During correlation of weak alerts, the aggregations are updated continuously. They are independent from the clustering, and they caused overhead of 7.84 and 8.90 seconds in the experiment

among 32 runs. The measured time includes database queries and updates. Although the clustering depends on the parameters *min_pts* and *eps*, the clustering overhead with DBSCAN for the final aggregations at the end of the experiment was almost constant between 0.06 and 0.08 seconds across 32 different parameter combinations.

This experiment furthermore analyzed the number of weak meta alerts depending on the clustering parameters, in particular for *min_eps* ranging from 1 to 10 and *eps* ranging from 0.1, i.e., less than 1, to 100. With higher *eps*, neighbor IP addresses with larger differences in alert count are grouped together, resulting in less clusters. A high value of $eps = 100$ dominates over the analyzed parameter range of *min_pts* and leads to between 134 and 139 weak meta alerts. In contrast, when choosing a low value for $eps = 0.1$, the influence of *min_pts* becomes effective. The resulting number of weak meta alerts then goes from 363 when $min\_pts = 1$ to 152 when $min\_pts = 10$.

This experiment indicates that filtering weak alerts and correlating them to weak meta alerts indeed points to APT steps that otherwise would go unnoticed. This is crucial for the detection of APT attacks as traditional alert correlation was shown to either neglect weak alerts at all (for the `PS-EXEC via SMB` attack), or falsely correlate them with unrelated alerts (for the `SQL & Web injection`).

6.6.3 Unfiltered Alerts Evaluation

In the previous experiment, the correlation performed on weak alerts. Intuitively, this approach relies on the traditional correlation function $\mathscr{K}$ to detect any obvious and high-volume attacks and to filter alerts as weak that do not seem to belong to those attacks. Although designed for the correlation of weak alerts, the correlation is potentially able to correlate also unfiltered alerts. Thus, the experiment here evaluates if the correlation of weak alerts is still possible when performing on the whole alert set that includes the weak ones. This is relevant if the correlation function $\mathscr{K}$ leaves many false positive weak alerts that have to be compensated by the correlation.

In this experiment, the weak alert correlation processes the original 250,521 alerts from the Zeek IDS. To avoid the few weak alerts from the APT attack to falsely be identified to be related with some of the other alerts, the DBSCAN clustering (cf. Section 5.2.2.3) is set particularly strict. Using the parameters $min_{pts} = 1$ and $eps = 0.1$ generates 4,218 weak meta alerts. Although the complete correlation took 11.8 minutes, it still processes the whole alert set faster than using GAC as correlation function $\mathscr{K}$.

Attack	*Label*	*Direction*	*Attackers*	*Victims*	# Alerts
Trojan Download	`OtM`	`Intranet` → `Internet`	172.16.0.1	10.99.99.189, 23.218.54.7	34
PS-EXEC via SMB	`OtO`	`Intranet` → `Intranet`	172.16.0.1	172.16.0.5	1
SQL & Web injection	`OtO`	`Intranet` → `Intranet`	172.16.0.5	172.16.0.10	100
		...			

Table 6.19: Weak meta alerts that reflect an APT step.

Three of the generated weak meta alerts for the whole alert set correspond to actual attacks in the ground truth. They are shown in Table 6.19. The weak meta alerts for the `PS-EXEC via SMB` attack and the `SQL & Web injection` accurately identified the respective alerts. According to the weak meta alert for the `Trojan Download`, the internal IP address 172.16.0.1 connected to two victims on the `Internet` during the attack. But the ground truth reveals that the Trojan was actually downloaded from only one of these IP addresses. However, in total two APT steps are accurately identified and a third one contains some false positive alerts.

In summary, the weak alert correlation also works on the complete alert set in general. This is indicated by the high sensitivity in this experiment, as the correlation was able to find the three APT steps for that Zeek IDS alerts exist. However, applied to the complete alert set, the correlation potentially results in an enormous number of weak meta alerts unrelated to APT attacks.

6.6.4 Summary of Weak Alert Correlation

An evaluation of weak alert correlation has been presented for the detection of slow and stealthy attack steps. These steps would be filtered by other alert clustering approaches because of the low and temporally dispersed alert volume.

The weak alert correlation has been evaluated using a running APT example with five steps over seven days, assembled from real-world traffic data. The Zeek IDS analyzed the traffic and reported 250,521 IDS alerts. Afterwards, regular alert clustering of the GAC approach (cf. Section 6.5) first processed them into 370 meta alerts, leaving 1,593 alerts unclustered. Some of the meta alerts already correspond to APT attack steps but two remained undetected. Working with the unclustered alerts, the weak alert correlation algorithm successfully detected these two remaining steps among the weak alerts. Especially the separation of network zones effectively make the generated weak meta alerts point to the crucial APT attack steps. But also applied to all alert data, the algorithm identified these two attack steps and one additional with some false positives. However, the correlation should be applied to weak alerts only to avoid a high number of false positive weak meta alerts.

In summary, reconstructing APT attacks requires to apply general intrusion detection and traditional alert correlation in the first place. Although the weak alert correlation algorithm alone can identify attack steps, it generates significantly more weak meta alerts compared to applying it to weak alerts only. Anyhow, the algorithm seems to appropriately complement traditional algorithms that fail to correlate alerts from stealthy attack steps. In contrast to the weak alert correlation that assembles temporally dispersed alerts, the collaborative detection by attack scenario for the detection of spatially dispersed attacks is evaluated in the next section.

6.7 Evaluation of Collaborative Attack Correlation

In case the local IDS alerts cover only a fraction of the full attack, a collaborative exchange of alert data with other IDSes might be required to merge respective alerts for establishing a view on the whole attack. To achieve an efficient identification of similar or potentially equal attacks in the first place, the motif-based correlation approach in Section 5.3.2 fingerprints the attacks' scenario regarding the relations among the hosts and the usage pattern of ports. By comparing

attacks and identifying matching attack scenarios on the basis of this small fingerprint, the set of exchanged IDS alerts is reduced to a smaller selection.

This evaluation of the motif-based attack correlation analyzes to which extent the motif-based scenario fingerprint can preserve the characteristics of the attack scenario. Furthermore, the evaluation investigates the benefits of this fingerprint regarding the reduction of the exchanged data volume in a collaborative detection setup.

This section revises of the conference paper [HWF19]. The section first evaluates the applicability and correctness of the motif-based fingerprint to identify similar attacks. Afterwards, this section evaluates the applicability for collaborative attack correlation on real-world data.

6.7.1 Classification Evaluation

This first evaluation part investigates the most important properties of the motif-based classification. It particular, the motif-based abstraction of attacks is required to be small and the fingerprinting of attacks to be characteristic for their scenarios. The fulfillment of the first requirement is given, because the data volume, i.e., size of all alert data can be magnitudes larger than the size of motif signatures. Thus, the following experiments investigate if a single motif signature is representative for all variants of an attack scenario. For that, motif signatures have to be very similar for attacks of the same attack scenario, denoted as *intra-class-similarity*. In addition, the *inter-class-similarity* denotes the similarity between attacks from different attack scenarios. It is required to be low such that different attack scenarios can be distinguished.

6.7.1.1 Scenario Data

Although the correlation algorithm operates on alerts, it is important to notice that the algorithm works with abstractions of attacks, i.e., meta alerts. Hence, the input to the correlation algorithm are alerts of specific attacks or alert clusters, respectively. To create synthetic attack data, this experiment generates alerts for a six different attack scenarios, each of them defining a pattern for the data generation. An instance of an attack is determined by its attack pattern and its attack size. The values for IP and ports are randomly chosen from the full IP and port range, respectively.

Attack Patterns The attack pattern in the experiments here fulfill two purposes: 1) Generating synthetic alerts of attacks for the evaluation of scenario classification and 2) defining reference scenarios used during classification. The names and characteristics of the six attack scenarios are as follows:

- A distributed denial-of-service (**DDoS**) attack is characterized by alerts that all share the same destination IP and port. Thus, multiple attackers target a specific host and service. This attack is parameterized by α alerts that are generated on average per attacker. Attackers use random source ports, which are reused in subsequent alerts with a probability p.

- A **Scan** attack is characterized by alerts that share the same attacker IP. Random source ports are used to scan for the same destination port on multiple target machines. On average α alerts are generated per target and the attacker reuse a source port with a probability p.

- A distributed scan (**D-Scan**) is similar to a Scan attack but with both multiple attackers and targets, e.g., when a Scan is coordinated by a botnet [Haa+16]. Then, tasks for scanning all targets are split among the attackers. Characteristic for this scenario is the ratio of attackers to targets, denoted as θ. Additionally, targets might be scanned multiple times, i.e., from multiple attackers. In this case, α alerts are generated per target.

- A **Worm** attack is characterized by alerts that all share the same destination port. Additionally, all hosts are attackers and target randomly μ of other hosts via random source ports.

- An exploration (**Expl**) attack is characterized by a single attacker that targets f hosts. Each compromised host serves as source for attacks on further hosts. Each compromised host targets f new hosts and all source and target ports are random.

- The pattern of a convergence (**Conv**) attacker is the opposite of an Exploration attack. The actual target is attacked by f hosts that themselves are attacked by f hosts and so on.

These six attack patterns are parameterized during the experiments as follows. Per source or target, $\alpha = 1.5$ alerts are generated. Ports are reused with a probability $p = 50\%$ where appropriate. In case of lateral movement in a network, it is done with a spread factor of $f = 5$, which means that in each step a new compromised host targets 5 new hosts. If a scenario is characterized by multiple attackers and targets, their ratio is $\theta = 0.5$, which means the same amount of attackers and targets. In the case of the worm scenario, each host attacks $\mu = 10\%$ of the other hosts.

Attack Variations In reality, not all attacks of the same attack scenario are equal with respect to their alerts. Of course, they differ in the actual IPs and ports. The concrete values, however, are not visible anymore in the motif signature of the attack. More interesting is the variation in the attack size. Also, clustered alerts of an attack can contain false positives which causes variations in the alerts of an attack.

In particular, attack variations are caused by generating attacks of different size, i.e., the **population** ψ, which is the number of hosts involved in the attack. The attack patterns define how many of the individual hosts are attackers, targets, or both. For example, in a DDoS attack of size 100, there would be one target and 99 attackers.

Anyhow, different data sets including attack variations are used throughout the evaluation of scenario classification. The data sets consist out of the alerts from particular attack instances, i.e., an instantiated attack pattern of a particular population size. Note, however, that two instances are probably never equal since the alert generation based on the pattern introduces some randomness.

6.7.1.2 Similarities of Scenario Classes

The first experiment investigates the intra-class similarities and inter-class similarities for attacks of the same size $\psi = 100$, i.e., number of hosts. For that, the data set is made up of 1000 attack instances for each of the six scenarios defined in Section 6.7.1.1. Then, the similarity among the attacks is measured. Please note that the attack patterns themselves inherent some randomness, so that two attacks of the same scenario differ in their alert data even if they are of equal size. Apart from different IPs and ports, the relation for who of the attackers target whom of the victims is chosen differently every time an attack instance is generated. More randomness is introduced, because of the generation parameter $\alpha = 1.5$ multiple alerts are generated for some attackers, which also differs every time.

	Lowest Intra-Class Similarity [%]	**Highest Inter-Class Similarity [%] (with)**	
DDoS	99.64	78.77	(Worm)
Scan	99.29	73.58	(Worm)
D-Scan	88.26	73.42	(Conv)
Worm	89.98	78.77	(DDoS)
Expl	92.65	73.00	(D-Scan)
Conv	91.52	73.42	(D-Scan)

Table 6.20: Similarities for attacks with 100 hosts for six scenarios, both for the same and for different scenarios.

Table 6.20 measures how different attacks from an individual scenario might be, i.e., the lowest intra-class similarity. The highest variation is measured among the D-Scan attacks (lowest similarity of 88.26%) and the most similar attacks from the same scenario are the DDoS and Scan attacks, each with a similarity of more than 99%. The table also measures the inter-class-similarities and reports the highest similarities per scenario in Table 6.20. There are some attack scenarios that share characteristics. The worm is similar to DDoS and Scan with 78.77% and 73.58%, respectively. The D-Scan is similar to Expl and Conv with 73.00% and 73.42%, respectively.

As the lowest intra-class similarity is always higher than the highest inter-class similarity, motifs are an appropriate abstraction for alert data to preserve the characteristics of attack scenarios. For attacks of the same size, the results indicate that the approach can correctly classify attack scenarios, both in sense of identifying the correct reference scenarios and detecting scenarios in unsupervised clustering.

6.7.1.3 Scaling with Attack Size

Apart from the question if the intra-class-similarity is always higher than the inter-class-similarity, the influence of the size of an attack is of interest. This is important because attacks can greatly differ in their sizes, i.e., number of hosts ψ. The motif-based classification is required to detect the attack scenario for an attack with 100 hosts but also for an attack with 1000 hosts. For that, the next experiment generates attacks for the six attack scenarios described in Section 6.7.1.1 with different attack sizes. For the generated attacks, the intra-class similarity or the inter-class

similarity is calculated pairwise, respectively, depending on if two attacks are from the same scenario or not. This experiment calculates these similarities for different data sets that differ in the range of population sizes. The population parameter ψ controls the range between the size of smallest and largest attack, i.e., how different the attacks are with respect to their size. The smallest attacks always encompass 100 host and the largest attacks are of sizes in the range of $100 \leq \psi \leq 1000$.

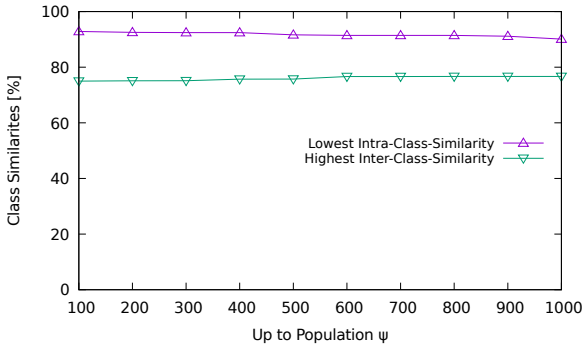


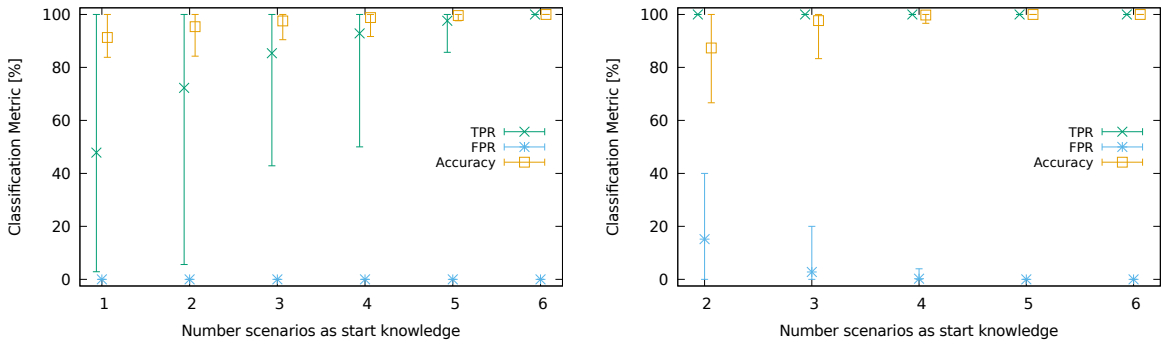
Figure 6.16: Distance between highest inter-class-similarity and lowest intra-class-similarity for attacks of several sizes. The range for attack sizes is $[100; \psi]$ and the similarities depend on the upper bound ψ.

Figure 6.16 plots the similarities depending on the attack sizes, i.e., when increasing the range of attack sizes in steps of 100. On the x-axis is the upper bound of the population size ψ, meaning a value on the y-axis depending on a specific ψ plots the similarities among attacks of sizes $[100; \psi]$ in steps of 100. On the y-axis is the lowest intra-class-similarity and highest inter-class-similarity among all attack scenarios. As long as the first curve is above the second one, it is possible to correctly classify the attacks in the respective data set. The gap between both curves indicates the potential range to choose the classification parameter τ from to achieve correct classifications. The results indicate that the motif-based abstraction can preserve the attack characteristics mostly independent from the attack size. For attacks of size 100 only, the width of the range for τ is 0.18. When clustering data sets that contain attacks of sizes between 100 and 1000, the width slowly decreases to 0.13. With respect to attack sizes between 100 and 1000 and with respect to the six attack scenarios in this experiment, the average value of τ should be about 0.83 +/- 0.07.

6.7.1.4 Learning new Scenario Classes

Another question, especially regarding how to choose τ, is how the accuracy of learning new scenarios depends on general knowledge of attack scenarios. While the unsupervised algorithm (cf. Section 5.3.2.3) can detect and characterize new scenarios, it is a matter of operating the attack clustering with an appropriate value for τ, not of specific previously defined reference signatures. For different choices of setting τ, this experiment shows how deriving unknown scenarios performs.

Although this evaluation defines six network-wide attack scenarios only, the motif-based classification is not limited to them. As the attack abstraction enables the identification and comparison of structural characteristics of attacks, the classification can potentially learn any new scenarios as long as they sufficiently differ in their communication structure. Another potential goal is to



(a) Based on lowest Intra-Class-Similarities. (b) Based on highest Inter-Class-Similarities.

Figure 6.17: Evaluation of attack clustering and classification for attacks of sizes between 100 and 1000. The clustering similarity τ and the resulting classification metrics depend on the selection of known scenarios as start knowledge.

divide known scenarios into more fine-grained ones, i.e., differentiate a reflection DDoS from a DDoS performed by a botnet. However, for simplicity this evaluation only defines the six generic network-wide scenarios here and leaves more (fine-grained) scenarios to future work.

According to the results from the previous experiment (cf. Section 6.7.1.3, the more τ is towards the upper bound of the possible range, the definition of attack scenarios becomes more strict. This results in a higher probability for false negatives in classifications in case the attacks of a new scenario have a higher variation than the previously known ones. However, a high τ also ensures precise classifications by reducing false positives in classifications in case the attacks of a new scenario share characteristics with a previously known one. If τ is set towards the lower bound of the range, there is a higher chance that attacks will be correctly classified although they look different than expected by the attack scenario. In turn, this increases the likelihood of false positives.

To investigate these expectations, the experiment here uses the same data set as in Section 6.7.1.3, containing attacks from the six scenarios with 100 to 1000 hosts. However, to investigate the learning of new scenarios, this experiment simulates a restricted start knowledge regarding the number of known scenarios, before applying unsupervised learning and evaluating the accuracy of the classes. This is repeated for each combination of 1 to 6 scenarios, for each combination measuring the lowest intra-class and highest inter-class similarities as in Table 6.20 and then determining the highest and lowest possible τ for the attack scenarios as in Figure 6.16. As there are multiple possible combinations of scenarios per number of scenarios, Figure 6.17 plots the minimum, maximum, and average value for the TPR, FPR, and accuracy of classification. As τ is based on the lowest known intra-class similarity in Figure 6.17a, attacks from all remaining scenarios will definitively go to another class, i.e., a lower TPR is accepted but therefore minimizing the FPR for learning new scenarios. In contrast, Figure 6.17b is based on the highest known inter-class similarity, which maximizes the TPR but accepts a higher FPR in return. Thus, choosing τ from the higher or lower boundary of possible range balances the ratio between expected TPR and FPR.

The results here for unsupervised clustering describe the worst-case performance of attack classification. For signature-based classification (cf. Section 5.3.2.2), attacks are only required to be more similar to the reference signature of their respective scenario than to the reference signature of a different scenario.

6.7.2 Real-World Evaluation

After analyzing the scenario classification on artificial data for which ground truth exists, the motif-based classification is applied on real-world data for the detection of attack scenarios. During this real-world evaluation, the set of reference scenarios consists out of one attack with 100 host for each of the six attack scenarios in Section 6.7.1.1. However, this evaluation here looks at both classifying real-world attacks with the help of reference scenarios and detecting scenarios with unsupervised clustering to compare them.

As the results of Section 6.7.1 indicate that the similarity threshold should be $\tau \in [0.76; 0.9]$, the respective range for τ is marked on the x-axis in all following figures. Choosing τ from this range is a prerequisite to distinguish attacks from the six reference scenarios.

6.7.2.1 Real-World Data

The real-world evaluation is using real-world data from the Internet Storm Center[8] that operates DShield[9], which is a platform for sharing data from security devices, e.g., from firewalls. The DShield logs consist of alerts from multiple sensors around the globe. The real-world experiments here are using all alerts collected on August, 22th in 2016. These are 4,517,497 alerts in total and are a result of several attacks.

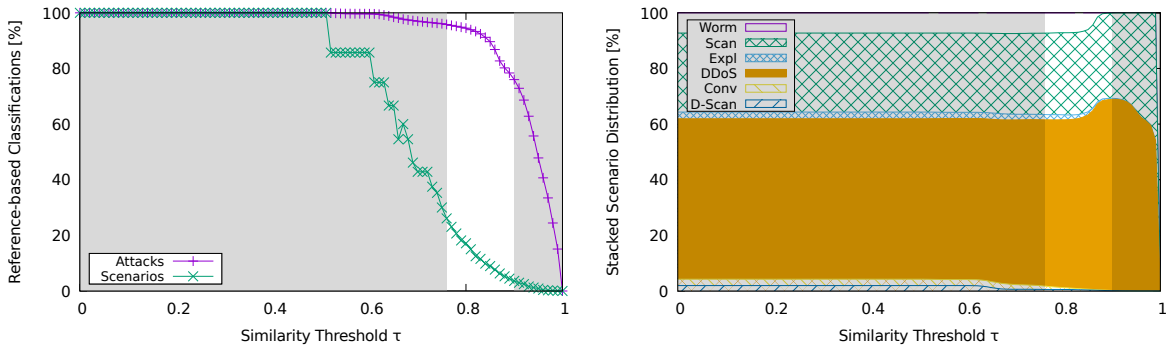
As motif-based scenario classification works on attacks, i.e., on alerts of the same attack, the DShield alerts are first grouped into clusters. For that, the alert clustering from GAC (cf. Section 6.5) is used that clusters alerts based on attribute similarity. In contrast to other clustering approaches, e.g., [ZLK09; Jul03], GAC does not enforce clusters with static attribute patterns. Instead, it identifies cliques of alerts that form a community, which allows a high diversity in the alert clusters and therefore in the attack scenarios. Clustering algorithms with static attribute patterns in contrast, would not be able to produce alert clusters for certain attack scenarios. Applying GAC clustering with a minimum similarity of 0.25 in between alerts and a clique size of 15 on the DShield data results in 34,204 clusters. They compose the data set of this real-world evaluation.

6.7.2.2 Efficiency of Motif Signatures

An efficient data structure with low overhead for the abstraction of attacks is necessary when sharing attack information and processing them in a distributed manner [Loc+05; YBJ04]. To evaluate the compression rate of the motif signatures for covering this requirement, this real-world experiment measures the data size of different potential sharing scenarios, i.e., alert representation structures, for the 34,204 attacks in the DShield data set.

For this evaluation of the compression rate, the experiment simulates the data exchange among the nodes in a CIDS if they would share their alert data. For that, the results indicate the data volume an individual node would send to others, when the locally monitored traffic equals the real-world data set.

8. SANS Technology Institute, Internet Storm Center, https://isc.sans.edu
9. https://secure.dshield.org



(a) Portion of attacks classified by reference scenarios and portion of identified reference scenarios among all identified scenarios.

(b) Portion of attacks per reference scenarios among all attacks classified by reference scenarios.

Figure 6.18: Classification of attacks using reference scenarios depending on similarity threshold τ.

Table 6.21 lists the data sizes for the alert data when exchanged with all attributes, with IP address and port only, and when only exchanging the motif signatures of attacks. Using the motif signatures to identify similar attacks, the size is reduced to 1.12% of the full alert data and 1.43% of the alerts with relevant attributes only.

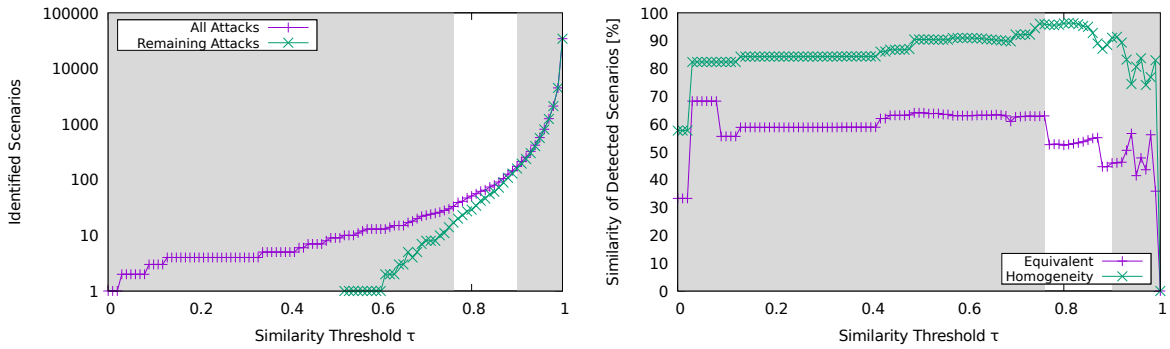
Exchanged Data	Data Size
Alerts with all attributes:	449 MB
Alerts with IP/Port only:	352 MB
Motif signatures of attacks:	5.1 MB

Table 6.21: Comparison of exchanged alert volumes.

6.7.2.3 Signature-based Classification

This experiment utilizes reference-based clustering to classify attacks in the DShield data set using the reference signatures from Section 6.7.1.1. The attacks are assigned one of the six reference scenarios based on the similarity threshold τ. For the attacks that could not be identifies as a reference scenario, unsupervised clustering classifies them, which results in additional classes, i.e., unknown attack scenarios.

Figure 6.18 shows the performance of the reference-based classification depending on τ. Figure 6.18a in particular illustrates how many attacks or scenarios have been classified or detected with the help of reference scenarios. For that, the curve labeled *attacks* plots the portion of the 34,204 attacks that have been assigned to one of the six reference scenarios. For a similarity threshold $\tau \leq 0.5$, all attacks are assigned a reference scenario. For larger τ, the attacks have to match the reference scenarios more closely. It is likely that the attacks obtained from the DShield data set come with false positive alerts. Therefore, these attacks cannot be assigned a reference scenario when close matches with the reference scenarios are required. Furthermore, the data set can contain attacks that are not covered by the six reference scenarios and will therefore not match any of them. In the marked range of τ, however, the signature-based classification identifies at least 76% and up to 96% of the attacks in the DShield data set.



(a) Detecting scenarios with unsupervised classification among all attacks and among remaining attacks from reference-based classification.

(b) Similarity between scenario classes detected by reference scenarios and detected by unsupervised clustering.

Figure 6.19: Identifying scenarios with reference-based and unsupervised classification depending on similarity threshold τ.

Furthermore, Figure 6.18a investigates the relation between the number of scenarios detected through reference-based clustering and the number of scenarios detected through unsupervised clustering of the remaining attacks. This plot also illustrates the portion of the total detected scenarios that are identified with the help of a reference scenario. In the marked range of τ, they are between 4% and 26%.

Figure 6.18b shows the distribution of reference scenarios among the identified attacks. The portions of the scenarios are stacked, so the aggregation of all six scenarios is 100%. As expected from a real-world data set, the most predominant attack scenarios are DDoS pattern with in between 60% and 69% as well as Scan pattern with in between 29% and 31%. Although for each of the six reference scenarios there is at least one attack identified in the marked range of τ, note that not all reference scenarios can technically show up in the DShield data set. This is because the destination IP addresses are hashed and therefore it is not possible to observe attacks in which an individual host is both, an attacker and a victim. This excludes the scenarios worm, expl, and conv. Considering this technical restriction, the conclusion is that in practice a large similarity threshold $\tau \leq 0.9$ should be chosen to avoid an unacceptable amount of false positive classifications.

6.7.2.4 Unsupervised Clustering

The last experiment evaluates the unsupervised clustering of the complete DShield real-world data set. For that, hierarchical clustering (cf. Section 5.3.2.3) is applied on the DShield attacks. Figure 6.19 reports the analysis of resulting classes depending on the similarity threshold τ.

Figure 6.19a counts the number of detected scenarios for unsupervised clustering on a log-scale. Within the marked range of τ, this results in between 33 and 174 clusters. Defining scenarios by higher similarities, i.e., τ, larger than 0.9, rapidly increases the number of scenarios and should only be used for fine-grained scenarios. In particular, two curves are plotted. The one labeled as *All Attacks* is the results for unsupervised clustering on the whole data set. In comparison, the curve labeled as *Remaining Attacks* refers to the results of unsupervised clustering on the attacks with unknown scenario from the previous experiment in Section 6.7.2.3. Altogether, both numbers of detected scenarios indicate that only searching for the six reference scenarios is

not enough. Instead, it is important to also apply unsupervised clustering at least on attacks for which no reference scenario can be assigned.

Furthermore, this experiment compares the results of reference-based and unsupervised clustering. The motivation is to find out for which value of the clustering parameter τ the two methods give consistent results. As no ground truth exists, two metrics are defined to measure the similarity between the reference clusters $C_i^r \in \hat{C}^r$ for respective reference signatures $R_i^r \in \hat{R}^r$ and the unsupervised clusters $C_i^u \in \hat{C}^u$:

- The metric **Equivalent** indicates how close candidates among the unsupervised clusters $C_i^u \in \hat{C}^u$ match the reference clusters $C_i^r \in \hat{C}^r$. For that, the experiment finds the best candidate cluster C_i^u for every reference cluster C_i^r based on the Jaccard index, i.e., intersection over union, which is calculated by $\frac{|C_i^u \cap C_i^r|}{|C_i^u \cup C_i^r|}$. An unsupervised cluster C_i^u can be matched to at most one reference cluster C_i^r. The metric represents the average Jaccard metric for the best matches among all reference clusters $\hat{C}^r$.
- The metric **Homogeneity** indicates the average accuracy for clusters $C_i^u \in \hat{C}^u$ that include at least one attack identified by the reference-based classification. In each of these clusters C_i^u, among all contained attacks, the experiment measures the fraction of attacks from the scenario that is present in the cluster most frequently. The metric represents the average homogeneity among all these clusters weighted by their sizes.

Figure 6.19b shows the comparison between signature-based clustering and unsupervised clustering according to the metrics *Equivalent* and *Homogeneity*, depending on the similarity threshold τ. For the marked range of τ, the *Homogeneity* is between 87% and 97% and the *Equivalent* is between 56% and 72%. According to the experiment results of Section 6.7.2.3, a similarity threshold τ close to 0.9 seems to be reasonable. Although the highest *Homogeneity* is achieved for $\tau = 0.81$, which is in the lower half of the possible values for τ, the *Homogeneity* at $\tau = 0.9$ is still at 91%. However, note that between $0.81 \leq \tau \leq 0.9$ the metric *Homogeneity* drops to 87%. This drop correlates with the changes of the proportions among the different attack scenarios in Figure 6.18b. From that perspective, $\tau = 0.9$ would be recommended to find attacks for the six scenario classes.

The two metrics *Homogeneity* and *Equivalent* for comparing signature-based clustering and unsupervised clustering indicate the following. Clustering attacks from unknown scenarios is done with high uncertainty. While unsupervised clustering is able to differentiate between attacks from different scenarios, it will not be able to perfectly classify a high variability of attacks without any previous knowledge. Hence, it is a good approach to provide as much reference scenarios as possible and to use unsupervised clustering to learn new attack scenarios and to create reference scenarios in a semi-supervised fashion.

6.7.3 Summary of Collaborative Attack Correlation

An evaluation of motif-based scenario classification has been presented for the collaborative identification of similar attacks. Instead of sending all alert data to every other node in a CIDS, the nodes can already identify candidates for attack matches based on comparing the motif-based scenario fingerprints of the attacks. Afterwards, nodes would still need to exchange raw IDS alerts, however, they can limit themselves to those alerts that belong to a potential match.

The correctness of classifying the attack scenario using the motif-based attack abstraction has been demonstrated by six attack patterns of network-wide attacks. The comparison of intra-class and inter-class similarities, i.e., similarities of attacks from the same or different scenarios, indicates that clustering attacks by their scenarios works, even for small attack variations that probably occur in real world. In particular, the analysis of classifying artificial attack data results in the recommendation to choose the clustering parameter τ in between 0.76 and 0.9.

The evaluation of the approach on real-world data details the analysis results. It further investigates the influence of the clustering parameter τ on the detected scenarios. In particular, the best performance could be achieved when clustering attacks with a similarity of around 90%. Apart from that, the experiment results highlight how to start with known attack scenarios and learn new attack scenarios in a semi-supervised fashion. It is best to start with as many reference scenarios as possible and to add scenarios from unsupervised clustering only when confirmed manually.

In summary, the motif-based scenario classification eases the comparison of large amounts of alert clusters. Converted into motif signatures, they are of small sizes and thus can be compared very fast. With the help of this abstraction, the approach identifies known attack scenarios, detects similar attacks, and can even learn about new attack scenarios.

6.8 Summary of Evaluation

This chapter has conducted a detailed evaluation of intrusion detection measures and algorithms that perform security monitoring and alert correlation. For the fulfillment of the requirements from Section 3.1, especially the detection accuracy, the measures are combined in a way such that they together detect various kind of network-wide attacks and summarize them in a concise intrusion report. For that, Section 6.1 has discussed the order in which the different measures and algorithms have to be placed during the data processing for interoperability reasons and to achieve effective intrusion detection. The evaluation of the partial measures and algorithms has been given in the subsequent sections.

The evaluation results of zeek-osquery in Section 6.3 indicate that network intrusion detection and in particular the security monitoring can leverage the fine-grained correlation with additional monitoring data from hosts. The result is an extended visibility through linking network flows with process information that enables insights into the semantics of network communication on the hosts. On that basis, zeek-osquery running in a testbed successfully detected and reconstructed the execution of malicious Internet files or SSH stepping-stones and made the hosts to assist in analyzing encrypted traffic. The underlying correlation platform is general purpose anyhow, and designed to scale even with large networks. This has been confirmed by stress tests in another testbed setup. Apart from that, a small real-world deployment has demonstrated the host-network correlation. The achieved network visibility through the attribution of network flows performs better by factor 1406 compared to other state-of-the-art approaches and reaches an attribution rate of over 96%. All these experiment results indicate that security monitoring with zeek-osquery leads to high-quality monitoring data.

Section 6.4 has evaluated another approach for intrusion detection on correlated monitoring data that requires network data only. Specifically for network- or Internet-wide attack scenarios, the relations and similarities among network flows are exploited to detect the scenarios' characteristics in the relations among communicating hosts. This has been demonstrated for the scenarios

distributed scan campaign and P2P botnets. Their detection is performed on the monitoring data from the network only. The evaluation of detecting scan campaigns in Section 6.4.1 has analyzed real-world Internet traffic for global campaigns. In addition to the detection results from backbone level, smaller networks have been simulated to compare their detection accuracy. Similarly, the botnet detection accuracy from the perspective of networks with different sizes has been compared. For that, Internet hosts have been simulated to be infected by a P2P botnet malware, based on real-world botnet communication graphs. The detection results for both scenarios indicate that their characteristics are not only apparent on global level, but also partly for individual networks. Particularly the botnet detection still works for a few infections within the network with high precision, because their P2P communication pattern significantly differs from other hosts. In contrast, large scan campaigns cannot be detected by very small networks alone but they have to cooperate with other network sites to cover a larger fraction of the communication.

When a network-wide attack cannot be directly detected at large from the monitoring data itself, the other common signature- or anomaly-based mechanisms of the IDS in place eventually report some alerts as indicators for an attack. It is the task of alert correlation to assemble these alerts to an intrusion report, as evaluated in Sections 6.5 to 6.7.

The evaluation of analyzing the alert clustering algorithm in GAC (cf. Section 6.5.1.2) indicates that especially distributed attacks with many alerts can effectively be assembled to attacks, without the need of narrowing down the clustering parameters too much. The performance on real-world alerts has been additionally demonstrated in Section 6.5.2.2. The resulting alert clusters include up to 99.3% of the real-world alerts. The remaining unclustered alerts could be false positives, i.e., not related to a larger attack, or they are more important than they seem at first sight. To clarify on these alerts, weak alert correlation checks if they eventually belong to a stealthy APT attack step. This correlation has been evaluated in Section 6.6 on an APT scenario constructed from real-world data traffic sets, including real APT malware but also other unrelated attacks. The results indicate that weak alert correlation complements the clustering approach for bulk attacks in GAC and is recommended to always be used in combination. This is because GAC already successfully detected some steps of the constructed APT attack. As less alerts are going into the weak alert correlation this way, the results can be as fine-grained as possible.

Before linking the alert clusters, i.e., attack steps, the attacks detected locally on sensors of a CIDS are supposed to be compared with the attacks from the other sensors in the network to identify global attacks and to merge the respective alert clusters. The motif-based approach proposed for this task has been evaluated in Section 6.7. The results indicate that the motif-based attack fingerprint preserves the scenario characteristics while enabling the reduction of data exchange volume by about 99%. The ability to match similar attacks on the basis of this fingerprint has been demonstrated by classifying attacks with six predefined scenarios but also by learning new scenarios.

Section 6.5.1.3 indicates that identifying the scenario context for the multi-step detection in GAC is robust against false positive alerts in the clusters. With the range of favorable certainty values derived from these analysis result, the real-world evaluation of the scenario context in Section 6.5.2.3 lets expect only few false positive context labels. Also the real-world results of the multi-step detection conducted in Section 6.5.2.4 are promising and indicate that utilizing the scenario context gives high-certainty reports about the relations of two cluster with overlapping IP addresses.

7 Conclusion

This chapter reviews the challenges of intrusion detection regarding attacks that are threatening the network at large and summarizes the contribution of this thesis. Thus, Section 7.1 presents the developed mechanisms in more detail and further discusses the main results of their evaluation. Finally, future work is sketched in Section 7.2.

7.1 Summary

Once an attacker infiltrated the computer network and gained foothold, the attack potentially affects the overall network operation regarding either a majority of hosts or particularly critical hosts. The detection of such network-wide attacks is required to be accurate to stop the attack entirely. Having an IDS in place alone is usually not sufficient because singleton IDS alerts only sparsely cover the full attack story. Instead, a detailed look at the scene is required all the time to fully capture an attack. Although a comprehensive monitoring is a fundamental step towards accurate intrusion detection, the malicious activities still need to be detected and also concisely summarized so that the full attack becomes apparent.

In this thesis, requirements beyond a traditional IDS have been derived to serve as a guideline for a discussion of the current state of the art. This discussion has revealed that most known intrusion detection approaches fail to include certain aspects of attacks in the scope of this thesis. Approaches for the detection of attacks with a lot of similar activity such as distributed attacks suffer from two problems. Either, they are not applicable to a broad range of attack scenarios in the first place such as the port scan detector SPICE [SHM02]. Or the approaches exclude any attack that is not following their definition of a malicious pattern, such as the root cause analysis on the basis of IDS alerts [Jul03]. Also the detection of intrusions that proceed far into the network via multiple steps is not fully provided by most of the analyzed approaches. Only two promising approaches for this kind of attacks have been identified. ZePro [Sun+16] reconstructs the most likely attack path through the network based on fine-grained operating system (OS) data and infection probabilities. HOLMES [Mil+19] identifies high-level advanced persistent threat (APT) patterns by transforming the fine-grained OS data into actions at an intermediate abstraction level and checks for their combinations to realize an APT attack. For this, however, particular expert knowledge about the patterns and APT steps is required. Apart from this, further approaches have been analyzed that incorporate various data in addition to the monitored data to assist in network intrusion detection. These approaches, however, are eventually helpful for manual threat-hunting but rarely conduct intrusion detection themselves, e.g., HERCULE [Pei+16] groups related activities from different logs – similarly to security information and event management (SIEM) systems – but without classifying the groups as malicious. At most, these approaches provide additional context to the intrusion detection. Nevertheless, the idea of supporting the network intrusion detection with additional context and correlated data has been adopted for this thesis and has influenced the development of measures for a more powerful IDS.

The detection system in this thesis has been developed as a result of the drawbacks of the current state of the art. The presented system encompasses several mechanisms that correlate information or derive additional context to enhance the detection. The foundation is a platform for the fine-grained correlation of monitoring data about hosts and their network communication. It extends the monitoring visibility in real-time so that detection algorithms and threat hunters can benefit from the joint host and network monitoring. The application of this correlation concept to the communication relations between hosts enables the detection of some distributed attacks such as port scan campaigns or peer-to-peer (P2P) botnets. The detection exploits scenario-specific characteristics that become apparent in the correlated communication relations. Furthermore, the detection system assembles and summarizes regular IDS alerts that are reported plentifully in daily operation to highlight attacks that spread into the network en masse or in depth. This alert correlation also overcomes the specific challenges that come with temporally and spatially dispersed alerts. Stealthy attacks or APTs with infrequent alerts are still detected even when they spread over a long time period. Additionally, the system efficiently identifies and brings together those alerts from different IDS sensors deployed in the network that belong to the same attack against the network at large. All given mechanisms have been evaluated extensively by simulations, on real-world data, in testbeds, and even in small real-world deployments. Several metrics have been developed during the different experiments in this thesis to measure each mechanism's contribution towards an accurate detection of intrusions that pose a threat to the network at large. The individual contributions of this thesis and their working along the intrusion detection process are described in the following.

As a prerequisite for accurate intrusion detection, *zeek-osquery* enhances the quality of monitoring data by a fine-grained correlation of host and network data. The additional context in form the semantics of hosts regarding their network communication compensates that network monitoring alone cannot capture all aspects of an attack in detail. The fine-grained correlation result is available for intrusion detection in real-time or is logged and used to trace back intrusions manually by threat hunters. Causally linking monitoring data across the host and network domain has shown to achieve an extended visibility that sheds light on the network communication. This not only allows to enforce fine-grained network policies on the basis of applications, but also enables the detection when malicious files that were downloaded from the Internet are executed on the hosts, among others. Apart from such specific use cases, the correlation platform has been designed for various kinds of correlations and analyzed to perform efficient and scalable also in large networks.

When the communicating hosts cannot be controlled to contribute to the host and network correlation, intrusion detection can still benefit from correlated network data. Correlating the communication of several hosts in the network enables the detection of some specific network-wide attack scenarios that are seen daily on the Internet. Their detection is based on scenario-specific characteristics that become apparent in the correlated monitoring data of network communication. This way, bots in a *P2P botnet* are detected by their characteristic communication pattern among each other and *scan campaigns* from different sources are detected by the characteristic scan activity. Although these attacks often have an Internet-wide scope, the experiments have shown that the detection not necessarily requires a global view on the Internet traffic. The scenario characteristics are partly apparent in local monitoring data so that network sites themselves can deploy the detection to some extent.

However, sometimes it is not directly apparent that a set of monitored activities in the network reflects an attack. Instead, certain events seem suspicious and are reported as alerts, indicating

that an attack is going on. Such alerts are reported by means of a regular IDS and equally important for intrusion detection along with the detection results from zeek-osquery or the detection on correlated network communication. Alert correlation, however, has to assemble these IDS alerts to the full picture of the attack, making the attack visible to its full extend.

Specifically for attacks in the scope on this thesis, the *graph-based alert correlation* (GAC) assembles alerts from two types of network-wide attacks. Although generally applicable, GAC focuses on attacks against multiple hosts at the same time or in sequence, i.e., distributed or multi-step attacks, respectively. The two-step approach first clusters alerts with similar attributes to distributed attacks. These resulting alert clusters are afterwards interconnected to detect multi-step attacks by overlapping clusters with same IP addresses. This second step of the approach makes use of the attack scenario that GAC derives from the relation among the hosts in an alert cluster. The evaluation indicates that the clustering covers a large variety of attacks because of the flexible definition of alert similarity. Furthermore, both the scenario context and the interconnection of attack steps has been evaluated to be robust against false positive alerts and to be accurate with high certainty.

In practice, alert correlation and also GAC suffers from two challenges, referred to as temporally and spatially dispersed alerts. They are the result from stealthy attack steps and attacks with a target scope beyond the monitored network, respectively. The one challenge is to identify that infrequent alerts belong together despite large time period between the occurrence of two alerts. The other challenge is that monitoring captures the attack only partly because it targets also other network sites the same way.

The challenge of temporally dispersed alerts from APT-like attacks is addressed by *weak alert correlation* that checks unclustered alerts to belong to a slow and ongoing attack step instead of just filtering them as irrelevant. APT characteristics are exploited when assembling such unclustered and weak alerts to APT attack steps. The experiments have shown an APT attack that cannot be detected by simply clustering alerts with GAC. Instead, weak alert correlation has to assemble candidates of APT alerts over a long time period to also detect the particularly stealthy attack steps that would have been gone unnoticed otherwise. The challenge of spatially dispersed alerts is addressed by *collaborative attack correlation*. It enables an efficient exchange and comparison of alert data to make several IDS sensors aware of that they captured parts of the same network-wide attack. Instead of exchanging all their alerts, the sensors fingerprint their locally observed attacks with respect to the attack's scenario and exchange these small attack fingerprint for an efficient comparison and identification of related alerts. The data exchange volume has been significantly reduced in the experiments while still being able to identify attacks that are of the same scenario type.

Even though the focus of the experiments has been to measure the contribution of each mechanism regarding a successful intrusion detection of network-wide attacks, different selections of mechanisms have been combined to conduct the experiments. Thus, the combination of all mechanisms as an end-to-end evaluation pipeline has not only been described in theory, they have also been selectively combined throughout their evaluation. Thus, the combination of security monitoring and alert correlation in the overall system implements intrusion detection for a variety of attacks that threaten the network at large.

7.2 Future Work

The possible future work in the context of this thesis and the related publications cover different areas of network intrusion detection.

Regarding the assistance of hosts in decrypting Transport Layer Security (TLS) connections, additional work is required to make the plaintext payload available for intrusion detection in real-time. So far, it has been demonstrated that TLS key material can be retrieved from the hosts and mapped to TLS connections in the network. While this is sufficient for forensic investigations on traffic captures, live decryption has to solve some additional challenges. These include that the encrypted communication is potentially already ongoing whereas the key for decryption has not been retrieved yet. Also, the decrypted messages should be analyzed transparently as it would be with unencrypted messages.

Furthermore, future work should enhance GAC to cluster alerts from long-running bulk attacks beyond the borders of alerts batches. While weak alert correlation solves the challenge of clustering infrequent alerts across multiple batches, alerts of the same attack filling multiple consecutive batches end up in one cluster per batch but should be merged to a single cluster to represent the attack. A solution for this could carry alert clusters in their representation as meta alert over to the next batch as long as the cluster gets extended with more alerts.

An APT attack detection needs to be developed that not only assembles respective alerts to APT-like attack steps but also interconnects them similarly to the GAC approach by overlapping IP addresses. The same way APT characteristics have been exploited to identify APT attack steps, their ensemble should also be checked for particular APT characteristics. On a positive match, the result would indicate that the attack is indeed an APT. Additional work could include an attacker model like the intrusion kill chain [HCA11] that requires the attack to proceed along different stages to take over the network or parts of it.

Moreover, additional future work is required regarding the detection of attack scenarios based on the correlated network communication. This thesis has demonstrated how to detect scan and botnet characteristics in large communication graphs. However, there are more scenarios like distributed denial-of-service (DDoS) or worm spreading that eventually can be detected in a similar way. While the characteristics of a worm spreading might be easily derived from the detection approach of P2P botnets, the DDoS is a network threatening attack that shows a fundamentally different communication pattern.

Building upon the similarity measure between two attacks based on their scenario fingerprints, additional work is required to incorporate this measure into an collaborative intrusion detection system (CIDS) [Vas+15b]. A distributed communication schema must be developed that allows a CIDS node to efficiently distribute new attack fingerprints to others.

Concluding, the detection system composed out of the presented contributions in this thesis provides solutions to most of the problems security operators are faced with when detecting attacks that are targeting the network at large. However, there is still room for further improvements. This covers additional work towards a live decryption of TLS session in zeek-osquery, concisely summarizing long-running bulk attacks, and detecting APT attacks in their characteristic context. Furthermore, future work is required to detect more network-wide attacks like DDoS or worm spreading by their scenario-specific characteristics in the correlated network data, and a communication schema is required to deploy the attack similarity based on scenario fingerprints in a distributed CIDS.

Bibliography

[Aba+03] Cristina Abad et al. *Log correlation for intrusion detection: A proof of concept*. In: *19th Annual Computer Security Applications Conference, 2003. Proceedings.* IEEE. 2003, pp. 255–264.

[Acc19] Accenture, Ponemon Institute. *The Cost of Cybercrime: Unlocking the Value of Improved Cybersecurity Protection*. Tech. rep. Accenture, 2019.

[AHS14] Johanna Amann, Seth Hall, and Robin Sommer. *Count me in: Viable distributed summary statistics for securing high-speed networks*. In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2014, pp. 320–340.

[AK13] Ahmed AlEroud and George Karabatis. *A system for cyber attack detection using contextual semantics*. In: *7th international conference on knowledge management in organizations: service and cloud computing*. Springer. 2013, pp. 431–442.

[AK14] Ahmed Aleroud and George Karabatis. *Context infusion in semantic link networks to detect cyber-attacks: a flow-based detection approach*. In: *2014 IEEE International Conference on Semantic Computing*. IEEE. 2014, pp. 175–182.

[AK17] Ahmed Aleroud and George Karabatis. *Contextual information fusion for intrusion detection: a survey and taxonomy*. In: *Knowledge and Information Systems* 52.3 (2017), pp. 563–619.

[AL01] Magnus Almgren and Ulf Lindqvist. *Application-integrated data collection for security monitoring*. In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2001, pp. 22–36.

[AMH16] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. *A survey of network anomaly detection techniques*. In: *Journal of Network and Computer Applications* 60 (2016), pp. 19–31.

[And+13] Dennis Andriesse et al. *Highly resilient peer-to-peer botnets are here: An analysis of gameover zeus*. In: *2013 8th International Conference on Malicious and Unwanted Software:" The Americas"(MALWARE)*. IEEE. 2013, pp. 116–123.

[ASU88] Alfred Aho, Ravi Sethi, and J Ullman. *Compilers: principles, techniques, and tools*. 1988.

[ATF09] Edward G Allan Jr, William H Turkett, and Errin W Fulp. *Using network motifs to identify application protocols*. In: *GLOBECOM 2009-2009 IEEE Global Telecommunications Conference*. IEEE. 2009, pp. 1–7.

[AZ09] Safaa O Al-Mamory and Hongli Zhang. *Intrusion detection alarms reduction using root cause analysis and clustering*. In: *Computer Communications* 32.2 (2009), pp. 419–430.

[BA99] Albert-László Barabási and Réka Albert. *Emergence of scaling in random networks*. In: *science* 286.5439 (1999), pp. 509–512.

[Bai+05] Michael Bailey et al. *The blaster worm: Then and now*. In: *IEEE Security & Privacy* 3.4 (2005), pp. 26–31.

[Bar+10] Rafael Ramos Regis Barbosa et al. *Simpleweb/University of Twente Traffic Traces Data Repository*. CTIT Technical Report Series TR-CTIT-10-19. Netherlands: Centre for Telematics and Information Technology (CTIT), 2010.

[Bar14] Sean Barnum. *Standardizing cyber threat intelligence information with the Structured Threat Information eXpression (STIX)*. Tech. rep. MITRE Corporation, Feb. 2014, p. 22. URL: https://stixproject.github.io/about/STIX_Whitepaper_v1.1.pdf (visited on 03/24/2020).

[Bat+15] Adam Bates et al. *Trustworthy Whole-System Provenance for the Linux Kernel*. In: *Proceedings of the 24th USENIX Conference on Security Symposium*. USA: USENIX Association, 2015, pp. 319–334.

[BBK11] Monowar H Bhuyan, Dhruba Kr Bhattacharyya, and Jugal K Kalita. *Surveying port scans and their detection methodologies*. In: *The Computer Journal* 54.10 (2011), pp. 1565–1581.

[BCF12] Matthew L Bringer, Christopher A Chelmecki, and Hiroshi Fujinoki. *A survey: Recent advances and future trends in honeypot research*. In: *International Journal of Computer Network and Information Security* 4.10 (2012), p. 63.

[BDA13] Elias Bou-Harb, Mourad Debbabi, and Chadi Assi. *Cyber scanning: a comprehensive survey*. In: *IEEE Communications Surveys & Tutorials* 16.3 (2013), pp. 1496–1519.

[Ben+09] Yoshua Bengio et al. *Learning deep architectures for AI*. In: *Foundations and trends® in Machine Learning* 2.1 (2009), pp. 1–127.

[BGA19] Debopam Bhattacherjee, Andrei Gurtov, and Tuomas Aura. *Watch your step! Detecting stepping stones in programmable networks*. In: *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE. 2019, pp. 1–7.

[BGD17] Norbert Blenn, Vincent Ghiëtte, and Christian Doerr. *Quantifying the spectrum of denial-of-service attacks through internet backscatter*. In: *Proceedings of the 12th International Conference on Availability, Reliability and Security*. ACM. 2017, p. 21.

[Bil+12] Leyla Bilge et al. *Disclosure: detecting botnet command and control servers through large-scale netflow analysis*. In: *Proceedings of the 28th Annual Computer Security Applications Conference*. 2012, pp. 129–138.

[Blo+08] Vincent D Blondel et al. *Fast unfolding of communities in large networks*. In: *Journal of statistical mechanics: theory and experiment* 2008.10 (2008), P10008.

[BLS13] Donabelle Baysa, Richard M Low, and Mark Stamp. *Structural entropy and metamorphic malware*. In: *Journal of computer virology and hacking techniques* 9.4 (2013), pp. 179–192.

[BMZ14] Sandeep Bhatt, Pratyusa K Manadhata, and Loai Zomlot. *The operational role of security information and event management systems*. In: *Security & Privacy* 12.5 (2014).

[BO04] Albert-Laszlo Barabasi and Zoltan N Oltvai. *Network biology: understanding the cell's functional organization*. In: *Nature reviews genetics* 5.2 (2004), pp. 101–113.

[Boh+17] Atul Bohara et al. *An unsupervised multi-detector approach for identifying malicious lateral movement*. In: *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*. IEEE. 2017, pp. 224–233.

[Bri+19] Robert A Bridges et al. *A survey of intrusion detection systems leveraging host data*. In: *ACM Computing Surveys (CSUR)* 52.6 (2019), pp. 1–35.

[Cai+05a] Min Cai et al. *Collaborative internet worm containment*. In: *IEEE Security & Privacy* 3.3 (2005), pp. 25–33.

[Cai+05b] Min Cai et al. *Fast and accurate traffic matrix measurement using adaptive cardinality counting*. In: *Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data*. 2005, pp. 205–206.

[Cai+05c] Min Cai et al. *Wormshield: Collaborative worm signature detection using distributed aggregation trees*. In: *Poster on the 2nd Symposium on Network System Design and Implementation (NSDI'05)*. 2005.

[CDH14] Ping Chen, Lieven Desmet, and Christophe Huygens. *A study on advanced persistent threats*. In: *IFIP International Conference on Communications and Multimedia Security*. Springer. 2014, pp. 63–72.

[CDM10] Baris Coskun, Sven Dietrich, and Nasir Memon. *Friends of an enemy: identifying local members of peer-to-peer botnets using mutual contacts*. In: *Proceedings of the 26th Annual Computer Security Applications Conference*. 2010, pp. 131–140.

[CDS14] Julie Connolly, Mark Davidson, and Charles Schmidt. *The Trusted Automated eXchange of Indicator Information (TAXII)*. Tech. rep. The MITRE Corporation, May 2014, p. 20. URL: http://taxii.mitre.org/about/documents/Introduction_to_TAXII_White_Paper_May_2014.pdf (visited on 03/24/2020).

[CER99] CERT Coordination Center (CERT/CC). *CERT incident note IN-99-07 distributed denial of service tools*. In: *http://www.cert.org/incident_notes/IN-99-07.html* (1999).

[Che+16] Ang Chen et al. *Dispersing asymmetric DDoS attacks with SplitStack*. In: *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. 2016, pp. 197–203.

[Chi+11] Shih-Chuan Chiu et al. *Incremental mining of closed inter-transaction itemsets over data stream sliding windows*. In: *Journal of Information Science* 37.2 (2011), pp. 208–220.

[CKT05] Weidong Cui, Randy H Katz, and Wai-tian Tan. *Design and implementation of an extrusion-based break-in detector for personal computers*. In: *21st Annual Computer Security Applications Conference (ACSAC'05)*. IEEE. 2005, 10–pp.

[CM02] Frédéric Cuppens and Alexandre Miege. *Alert correlation in a cooperative intrusion detection framework*. In: *Proceedings 2002 IEEE symposium on security and privacy*. IEEE. 2002, pp. 202–215.

[CO00] Frédéric Cuppens and Rodolphe Ortalo. *Lambda: A language to model a database for detection of attacks*. In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2000, pp. 197–216.

[Coh95] William W Cohen. *Fast effective rule induction*. In: *Machine learning proceedings 1995*. Elsevier, 1995, pp. 115–123.

[Cup01] Frédéric Cuppens. *Managing alerts in a multi-intrusion detection environment*. In: *Seventeenth Annual Computer Security Applications Conference*. 2001.

[Dai+15] Alberto Dainotti et al. *Analysis of a "/0" Stealth Scan from a Botnet*. In: *IEEE/ACM Transactions on Networking (TON)* 23.2 (2015).

[DC02] Oliver Dain and Robert K Cunningham. *Fusing a heterogeneous alert stream into scenarios*. In: *Applications of Data Mining in Computer Security*. Springer, 2002, pp. 103–122.

[DCF07] Herve Debar, David Curry, and Benjamin Feinstein. *The intrusion detection message exchange format (IDMEF)*. In: *Request for Comments (RFC)* 4765 (2007).

[DD00] John E Dickerson and Julie A Dickerson. *Fuzzy network profiling for intrusion detection*. In: *PeachFuzz 2000. 19th International Conference of the North American Fuzzy Information Processing Society-NAFIPS (Cat. No. 00TH8500)*. IEEE. 2000, pp. 301–306.

[Dee89] S. Deering. *RFC 1112: Host Extensions for IP Multicasting*. 1989.

[DM09] George Danezis and Prateek Mittal. *SybilInfer: Detecting sybil nodes using social networks.* In: *NDSS*. San Diego, CA. 2009, pp. 1–15.

[Dre+05] Holger Dreger et al. *Enhancing the accuracy of network-based intrusion detection with host-based context*. In: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer. 2005, pp. 206–221.

[DW01] Hervé Debar and Andreas Wespi. *Aggregation and correlation of intrusion-detection alerts*. In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2001, pp. 85–103.

[Edd07] W. Eddy. *RFC 4987: TCP SYN Flooding Attacks and Common Mitigations*. 2007.

[Eis+89] Ted Eisenberg et al. *The Cornell commission: on Morris and the worm*. In: *Communications of the ACM* 32.6 (1989), pp. 706–709.

[Esk+02] Eleazar Eskin et al. *A geometric framework for unsupervised anomaly detection*. In: *Applications of data mining in computer security*. Springer, 2002, pp. 77–101.

[Esk00] Eleazar Eskin. *Anomaly Detection over Noisy Data using Learned Probability Distributions*. In: *Proceedings of the Seventeenth International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc. 2000, pp. 255–262.

[Est+04] Cristian Estan et al. *Building a better NetFlow*. In: *ACM SIGCOMM Computer Communication Review* 34.4 (2004), pp. 245–256.

[FA16] Fatemeh Faraji Daneshgar and Maghsoud Abbaspour. *Extracting fuzzy attack patterns using an online fuzzy adaptive alert correlation framework*. In: *Security and Communication Networks* 9.14 (2016), pp. 2245–2260.

[Faw+16] Ahmed Fawaz et al. *Lateral movement detection using distributed data fusion*. In: *2016 IEEE 35th Symposium on Reliable Distributed Systems (SRDS)*. IEEE. 2016, pp. 21–30.

[FBD15] Claude Fachkha, Elias Bou-Harb, and Mourad Debbabi. *Inferring distributed reflection denial of service attacks from darknet*. In: *Computer Communications* 62 (2015), pp. 59–71.

[Fei+03] Laura Feinstein et al. *Statistical approaches to DDoS attack detection and response*. In: *Proceedings DARPA information survivability conference and exposition*. Vol. 1. IEEE. 2003, pp. 303–314.

[Flo62] Robert W Floyd. *Algorithm 97: shortest path*. In: *Communications of the ACM* 5.6 (1962), p. 345.

[FMF14] Romain Fontugne, Johan Mazel, and Kensuke Fukuda. *Hashdoop: A MapReduce framework for network anomaly detection*. In: *2014 IEEE conference on computer communications workshops (INFOCOM WKSHPS)*. IEEE. 2014, pp. 494–499.

[FMN05] Glenn A Fink, Paul Muessig, and Chris North. *Visual correlation of host processes and network traffic*. In: *IEEE Workshop on Visualization for Computer Security, 2005.(VizSEC 05).* IEEE. 2005, pp. 11–19.

[Fon+10] Romain Fontugne et al. *MAWILab: Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking*. In: *Proceedings of the 6th International COnference*. Co-NEXT '10. ACM, 2010. DOI: 10.1145/1921168.1921179.

[For18] Fortinet. *Threat Landscape Report Q3 2018*. Tech. rep. 2018. URL: https://www.fortinet.com/content/dam/fortinet/assets/threat-reports/threat-report-q3-2018.pdf (visited on 06/29/2020).

[Fou20] OpenSSL Software Foundation. *OpenSSL 3.0 - OpenSSLWiki*. 2020. URL: https://wiki.openssl.org/index.php/OpenSSL_3.0#Other_major_new_features (visited on 06/02/2020).

[Fra+11] Jerome Francois et al. *Botcloud: Detecting botnets using mapreduce*. In: *2011 IEEE International Workshop on Information Forensics and Security*. IEEE. 2011, pp. 1–6.

[FS11] Kevin R Fall and W Richard Stevens. *TCP/IP illustrated, volume 1: The protocols*. addison-Wesley, 2011. ISBN: 978-0321336316.

[FS16] Gianluigi Folino and Pietro Sabatino. *Ensemble based collaborative and distributed intrusion detection systems: A survey*. In: *Journal of Network and Computer Applications* 66 (2016), pp. 1–16.

[FWE+11] Jérôme François, Shaonan Wang, Thomas Engel, et al. *BotTrack: tracking botnets using NetFlow and PageRank*. In: *International Conference on Research in Networking*. Springer. 2011, pp. 1–14.

[Gat09] Carrie Gates. *Coordinated Scan Detection*. In: *NDSS*. 2009.

[GG15] Mohammad GhasemiGol and Abbas Ghaemi-Bafghi. *E-correlator: an entropy-based alert correlation system*. In: *Security and Communication Networks* 8.5 (2015), pp. 822–836.

[GN08] Gérard Govaert and Mohamed Nadif. *Block clustering with Bernoulli mixture models: Comparison of different approaches*. In: *Computational Statistics & Data Analysis* 52.6 (2008), pp. 3233–3245.

[Gök+14] Enes Göktas et al. *Out of control: Overcoming control-flow integrity*. In: *2014 IEEE Symposium on Security and Privacy*. IEEE. 2014, pp. 575–589.

[Gri+03] John L Griffin et al. *On the feasibility of intrusion detection inside workstation disks*. Tech. rep. School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 2003.

[GRS06] Debin Gao, Michael K Reiter, and Dawn Song. *Behavioral distance measurement using hidden markov models*. In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2006, pp. 19–40.

[GW97] Tal Grossman and Avishai Wool. *Computational experience with approximation algorithms for the set covering problem*. In: *European Journal of Operational Research* 101.1 (1997), pp. 81–92.

[GZC14] Sebastián García, Alejandro Zunino, and Marcelo Campo. *Survey on network-based botnet detection methods*. In: *Security and Communication Networks* 7.5 (2014), pp. 878–903.

[Haa+16] Steffen Haas et al. *On the resilience of P2P-based botnet graphs*. In: *2016 IEEE Conference on Communications and Network Security (CNS)*. IEEE. 2016, pp. 225–233.

[Har+16] Christopher R Harshaw et al. *Graphprints: Towards a graph analytic method for network anomaly detection*. In: *Proceedings of the 11th Annual Cyber and Information Security Research Conference*. 2016, pp. 1–4.

[HB06] Greg Hoglund and James Butler. *Rootkits: subverting the Windows kernel*. Addison-Wesley Professional, 2006. ISBN: 978-0321294319.

[HCA11] Eric M Hutchins, Michael J Cloppert, and Rohan M Amin. *Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains*. In: *Leading Issues in Information Warfare & Security Research* 1.1 (2011), p. 80.

[HCC92] Jiawei Han, Yandong Cai, and Nick Cercone. *Knowledge discovery in databases: An attribute-oriented approach*. In: *VLDB*. Vol. 18. 1992, pp. 574–559.

[HCC93] Jiawei Han, Yandong Cai, and Nick Cercone. *Data-driven discovery of quantitative rules in relational databases*. In: *IEEE transactions on Knowledge and Data Engineering* 5.1 (1993), pp. 29–40.

[HCL05] Kai Hwang, Ying Chen, and Hua Liu. *Defending distributed systems against malicious intrusions and network anomalies*. In: *19th IEEE International Parallel and Distributed Processing Symposium*. IEEE. 2005, 8–pp.

[Hel+03] Katherine Heller et al. *One class support vector machines for detecting anomalous windows registry accesses*. In: *Third IEEE International Conference on Data Mining (Workshop on Data Mining for Computer Security)*. 2003.

[HF18] Steffen Haas and Mathias Fischer. *GAC: graph-based alert correlation for the detection of distributed multi-step attacks*. In: *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. 2018, pp. 979–988.

[HF19] Steffen Haas and Mathias Fischer. *On the alert correlation process for the detection of multi-step attacks and a graph-based realization*. In: *ACM SIGAPP Applied Computing Review* 19.1 (2019), pp. 5–19.

[HFS98] Steven A Hofmeyr, Stephanie Forrest, and Anil Somayaji. *Intrusion detection using sequences of system calls*. In: *Journal of computer security* 6.3 (1998), pp. 151–180.

[Hil90] Mark D Hill. *What is scalability?* In: *ACM SIGARCH Computer Architecture News* 18.4 (1990), pp. 18–21.

[Hol+08] Thorsten Holz et al. *Measurements and Mitigation of Peer-to-Peer-Based Botnets: A Case Study on Storm Worm*. In: *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*. LEET'08. San Francisco, California: USENIX Association, 2008.

[HSF20] Steffen Haas, Robin Sommer, and Mathias Fischer. *zeek-osquery: Host-Network Correlation for Advanced Monitoring and Intrusion Detection*. In: *ICT Systems Security and Privacy Protection*. Springer, 2020.

[Hwa+05] Kai Hwang et al. *GridSec: trusted grid computing with security binding and self-defense against network worms and DDoS attacks*. In: *International Conference on Computational Science*. Springer. 2005, pp. 187–195.

[HWF19] Steffen Haas, Florian Wilkens, and Mathias Fischer. *Efficient Attack Correlation and Identification of Attack Scenarios based on Network-Motifs*. In: *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*. IEEE. 2019, pp. 1–11.

[HWF20] Steffen Haas, Florian Wilkens, and Mathias Fischer. *Scan Correlation – Revealing distributed scan campaigns*. In: *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE. 2020.

[JD02] Klaus Julisch and Marc Dacier. *Mining intrusion detection alarms for actionable knowledge*. In: *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2002, pp. 366–375.

[Jen87] Kurt Jensen. *Coloured petri nets*. In: *Petri nets: central models and their properties*. Springer, 1987, pp. 248–299.

[JK11] Krzysztof Juszczyszyn and Grzegorz Kołaczek. *Motif-Based Attack Detection in Network Communication Graphs*. In: *Communications and Multimedia Security*. Springer, 2011, pp. 206–213.

[JKR02] Jaeyeon Jung, Balachander Krishnamurthy, and Michael Rabinovich. *Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites*. In: *Proceedings of the 11th international conference on World Wide Web*. 2002, pp. 293–304.

[Joh67] Stephen C Johnson. *Hierarchical clustering schemes*. In: *Psychometrika* 32.3 (1967).

[Jul03] Klaus Julisch. *Clustering intrusion detection alarms to support root cause analysis*. In: *ACM transactions on information and system security (TISSEC)* 6.4 (2003), pp. 443–471.

[Jun+04] Jaeyeon Jung et al. *Fast portscan detection using sequential hypothesis testing*. In: *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*. IEEE. 2004, pp. 211–225.

[Kar+14] Shankar Karuppayah et al. *On advanced monitoring in resilient and unstructured P2P botnets*. In: *2014 IEEE International Conference on Communications (ICC)*. IEEE. 2014, pp. 871–877.

[Kar72] Richard M Karp. *Reducibility among combinatorial problems*. In: *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[KB96] Darren R Kerr and Barry L Bruins. *Network flow switching and flow data export*. US Patent 6,243,667. 1996.

[KC03] Samuel T King and Peter M Chen. *Backtracking intrusions*. In: *Proceedings of the nineteenth ACM symposium on Operating systems principles*. 2003, pp. 223–236.

[KDH16] Tiina Kovanen, Gil David, and Timo Hämäläinen. *Survey: Intrusion detection systems in encrypted traffic*. In: *Internet of Things, Smart Spaces, and Next Generation Networks and Systems*. Springer, 2016, pp. 281–293.

[Kee+16] Nathan Keegan et al. *A survey of cloud-based network intrusion detection analysis*. In: *Human-centric Computing and Information Sciences* 6.1 (2016), p. 19.

[Kin+05] Samuel T King et al. *Enriching Intrusion Alerts Through Multi-Host Causality*. In: *NDSS*. 2005.

[KJL11] Ryan KL Ko, Peter Jagadpramana, and Bu Sung Lee. *Flogger: A file-centric logger for monitoring file access and transfers within cloud computing environments*. In: *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE. 2011, pp. 765–771.

[Kru+03a] Christopher Kruegel et al. *Bayesian event classification for intrusion detection*. In: *19th Annual Computer Security Applications Conference, 2003. Proceedings.* IEEE. 2003, pp. 14–23.

[Kru+03b] Christopher Kruegel et al. *On the detection of anomalous system call arguments*. In: *European Symposium on Research in Computer Security*. Springer. 2003, pp. 326–343.

[Kru+05] Christopher Kruegel et al. *Polymorphic worm detection using structural information of executables*. In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2005, pp. 207–226.

[KS14] Ratinder Kaur and Maninder Singh. *A survey on zero-day polymorphic worm detection techniques*. In: *IEEE Communications Surveys & Tutorials* 16.3 (2014), pp. 1520–1549.

[KS94] Gene H Kim and Eugene H Spafford. *The design and implementation of tripwire: A file system integrity checker*. In: *Proceedings of the 2nd ACM Conference on Computer and Communications Security*. 1994, pp. 18–29.

[KT03] Christopher Kruegel and Thomas Toth. *Using decision trees to improve signature-based intrusion detection*. In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2003, pp. 173–191.

[Kum07] Sanjeev Kumar. *Smurf-based distributed denial of service (ddos) attack amplification in internet*. In: *Second International Conference on Internet Monitoring and Protection (ICIMP 2007)*. IEEE. 2007, pp. 25–25.

[Kwo+19] Donghwoon Kwon et al. *A survey of deep learning-based network anomaly detection*. In: *Cluster Computing* (2019), pp. 1–13.

[Lip15] Steven B Lipner. *The birth and death of the orange book*. In: *IEEE Annals of the History of Computing* 37.2 (2015), pp. 19–31.

[Llo82] Stuart Lloyd. *Least squares quantization in PCM*. In: *IEEE transactions on information theory* 28.2 (1982), pp. 129–137.

[LM04] Ling Li and Constantine N Manikopoulos. *Windows NT one-class masquerade detection*. In: *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004.* IEEE. 2004, pp. 82–87.

[Loc+05] Michael E Locasto et al. *Towards collaborative security and p2p intrusion detection*. In: *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*. IEEE. 2005, pp. 333–339.

[LRS03] Cynthia Bailey Lee, Chris Roedel, and Elena Silenok. *Detection and characterization of port scan attacks*. In: *Univeristy of California, Department of Computer Science and Engineering* (2003).

[LSM99] Wenke Lee, Salvatore J Stolfo, and Kui W Mok. *A data mining framework for building intrusion detection models*. In: *Proceedings of the 1999 IEEE Symposium on Security and Privacy (Cat. No. 99CB36344)*. IEEE. 1999, pp. 120–132.

[LX00] Wenke Lee and Dong Xiang. *Information-theoretic measures for anomaly detection*. In: *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*. IEEE. 2000, pp. 130–143.

[LZX13] Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. *High Accuracy Attack Provenance via Binary-based Execution Partition.* In: *NDSS*. 2013.

[Man+00] Stefanos Manganaris et al. *A data mining analysis of RTID alarms*. In: *Computer Networks* 34.4 (2000), pp. 571–577.

[McH00] John McHugh. *Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory*. In: *ACM Transactions on Information and System Security (TISSEC)* (2000).

[MD03] Benjamin Morin and Hervé Debar. *Correlation of intrusion symptoms: an application of chronicles*. In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2003, pp. 94–112.

[Mei04] Michael Meier. *A model for the semantics of attack signatures in misuse detection systems*. In: *International Conference on Information Security*. Springer. 2004, pp. 158–169.

[Mil+02] R. Milo et al. *Network Motifs: Simple Building Blocks of Complex Networks*. In: *Science* 298.5594 (2002), pp. 824–827.

[Mil+19] Sadegh M Milajerdi et al. *Holmes: real-time apt detection through correlation of suspicious information flows*. In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2019, pp. 1137–1152.

[Mir18] Ali H Mirza. *Computer network intrusion detection using various classifiers and ensemble learning*. In: *2018 26th Signal Processing and Communications Applications Conference (SIU)*. IEEE. 2018, pp. 1–4.

[MIT] MITRE. *CAPAC – Common Attack Pattern Enumeration and Classification (CAPEC)*. URL: https://capec.mitre.org (visited on 02/26/2020).

[MMS16] K Narasimha Mallikarjunan, K Muthupriya, and S Mercy Shalinie. *A survey of distributed denial of service attack*. In: *2016 10th International Conference on Intelligent Systems and Control (ISCO)*. IEEE. 2016, pp. 1–6.

[Moo+03] David Moore et al. *Inside the slammer worm*. In: *IEEE Security & Privacy* 1.4 (2003), pp. 33–39.

[Mor+02] Benjamin Morin et al. *M2D2: A formal data model for IDS alert correlation*. In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2002, pp. 115–137.

[MSK05] Michael Meier, Sebastian Schmerl, and Hartmut Koenig. *Improving the efficiency of misuse detection*. In: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer. 2005, pp. 188–205.

[Muh+18] Dominik Muhs et al. *On the Robustness of Random Walk Algorithms for the Detection of Unstructured P2P Botnets*. In: *2018 11th International Conference on IT Security Incident Management & IT Forensics (IMF)*. IEEE. 2018, pp. 3–14.

[Mun+09] Kiran-Kumar Muniswamy-Reddy et al. *Layering in provenance systems*. In: *Proceedings of the 2009 USENIX Annual Technical Conference (USENIX'09)*. USENIX Association. 2009.

[MV13] Fragkiskos D Malliaros and Michalis Vazirgiannis. *Clustering and community detection in directed networks: A survey*. In: *Physics Reports* 533.4 (2013), pp. 95–142.

[MZX16] Shiqing Ma, Xiangyu Zhang, and Dongyan Xu. *Protracer: Towards Practical Provenance Tracing by Alternating Between Logging and Tainting*. In: *Network and Distributed System Security Symposium (NDSS)*. 2016.

[Nag+10] Shishir Nagaraja et al. *BotGrep: Finding P2P Bots with Structured Graph Analysis*. In: *USENIX security symposium*. Vol. 10. 2010, pp. 95–110.

[Nar+14] Pratik Narang et al. *Peershark: detecting peer-to-peer botnets by tracking conversations*. In: *2014 IEEE Security and Privacy Workshops*. IEEE. 2014, pp. 108–115.

[NCR02] Peng Ning, Yun Cui, and Douglas S Reeves. *Constructing attack scenarios through correlation of intrusion alerts*. In: *Proceedings of the 9th ACM Conference on Computer and Communications Security*. 2002, pp. 245–254.

[Nia+20] Amirreza Niakanlahiji et al. *ShadowMove: A Stealthy Lateral Movement Strategy*. In: *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 2020.

[Nin+04] Peng Ning et al. *Building Attack Scenarios through Integration of Complementary Alert Correlation Method.* In: *NDSS*. Vol. 4. 2004, pp. 97–111.

[Nis+18] Antonia Nisioti et al. *From intrusion detection to attacker attribution: A comprehensive survey of unsupervised methods*. In: *IEEE Communications Surveys & Tutorials* 20.4 (2018), pp. 3369–3388.

[NS05] James Newsome and Dawn Xiaodong Song. *Dynamic Taint Analysis for Automatic Detection, Analysis, and SignatureGeneration of Exploits on Commodity Software.* In: *NDSS*. Vol. 5. Citeseer. 2005, pp. 3–4.

[OBr16] Dick O'Brien. *Dridex: Tidal waves of spam pushing dangerous financial Trojan*. Tech. rep. 2016. URL: https://www-west.symantec.com/content/dam/symantec/docs/security-center/white-papers/dridex-financial-trojan-16-en.pdf (visited on 05/11/2020).

[OGA05] Xinming Ou, Sudhakar Govindavajhala, and Andrew W Appel. *MulVAL: A Logic-based Network Security Analyzer.* In: *USENIX security symposium*. Vol. 8. Baltimore, MD. 2005, pp. 113–128.

[ONe+16] Mark O'Neill et al. *TLS proxies: Friend or foe?* In: *Proceedings of the 2016 Internet Measurement Conference*. 2016, pp. 551–557.

[Ort19] Felix C. Ortmann. *Temporal and Spatial Alert Correlation for the Detection of Advanced Persistent Threats*. Master Thesis. Universität Hamburg, July 2019.

[Pal+05] Gergely Palla et al. *Uncovering the overlapping community structure of complex networks in nature and society*. In: *nature* 435.7043 (2005), pp. 814–818.

[Pat+04] Swapnil Patil et al. *I3FS: An In-Kernel Integrity Checker and Intrusion Detection File System*. In: *LISA*. Vol. 4. 1. 2004, pp. 67–78.

[Pat+09] Abhinav Pathak et al. *Botnet spam campaigns can be long lasting: evidence, implications, and analysis*. In: *ACM SIGMETRICS Performance Evaluation Review* 37.1 (2009), pp. 13–24.

[Pax98] Vern Paxson. *Bro: a system for detecting network intruders in real-time*. In: *Proceedings of the 7th USENIX Security Symposium*. USENIX Association, 1998.

[Pax99] Vern Paxson. *Bro: a system for detecting network intruders in real-time*. In: *Computer Networks* 31.23 (1999), pp. 2435–2463. DOI: 10.1016/S1389-1286(99)00112-7.

[Pea85] Judea Pearl. *Bayesian networks: A model of self-activated memory for evidential reasoning*. In: *Proceedings of the 7th Conference of the Cognitive Science Society, University of California, Irvine, CA, USA*. 1985, pp. 15–17.

[Pea88] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988. ISBN: 1558604790.

[Pei+16] Kexin Pei et al. *Hercule: Attack story reconstruction via community discovery on correlated log graph*. In: *Proceedings of the 32Nd Annual Conference on Computer Security Applications*. 2016, pp. 583–595.

[PFV02] Phillip A Porras, Martin W Fong, and Alfonso Valdes. *A mission-impact-based approach to INFOSEC alarm correlation*. In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2002, pp. 95–114.

[Pie04] Tadeusz Pietraszek. *Using adaptive alert classification to reduce false positives in intrusion detection*. In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2004, pp. 102–124.

[PN97] Phillip A Porras and Peter G Neumann. *EMERALD: Event monitoring enabling response to anomalous live disturbances*. In: *Proceedings of the 20th national information systems security conference*. Vol. 3. 1997, pp. 353–365.

[Pos+81] Jon Postel et al. *RFC 793: Transmission control protocol*. 1981.

[QL03] Xinzhou Qin and Wenke Lee. *Statistical causality analysis of infosec alert data*. In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2003, pp. 73–93.

[QL04] Xinzhou Qin and Wenke Lee. *Attack plan recognition and prediction using causal networks*. In: *20th Annual Computer Security Applications Conference*. IEEE. 2004, pp. 370–379.

[Rab81] Michael O Rabin. *Fingerprinting by random polynomials*. Tech. rep. 1981.

[RCM11] Sebastian Roschke, Feng Cheng, and Christoph Meinel. *A new alert correlation algorithm based on attack graph*. In: *Computational intelligence in security for information systems*. Springer, 2011, pp. 58–67.

[RD13] Christian Rossow and Christian J Dietrich. *Provex: Detecting botnets with encrypted command and control channels*. In: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer. 2013, pp. 21–40.

[Reg19] Marvin Regorz. *Port Scan Alert Correlation*. Bachelor Thesis. Universität Hamburg, Sept. 2019.

[RN95] Stuart J Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Upper Saddle River, New Jersey, USA: Prentice Hall, 1995.

[Ros14] Christian Rossow. *Amplification Hell: Revisiting Network Protocols for DDoS Abuse.* In: *NDSS*. 2014.

[Rot+17] Rahmtin Rotabi et al. *Detecting strong ties using network motifs*. In: *Proceedings of the 26th International Conference on World Wide Web Companion*. 2017, pp. 983–992.

[Rud+16] Ethan M Rudd et al. *A survey of stealth malware attacks, mitigation measures, and steps toward autonomous open world solutions*. In: *IEEE Communications Surveys & Tutorials* 19.2 (2016), pp. 1145–1172.

[RVE18] Jos van Roosmalen, Harald Vranken, and Marko van Eekelen. *Applying deep learning on packet flows for botnet detection*. In: *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. 2018, pp. 1629–1636.

[Sch+00] Matthew G Schultz et al. *Data mining methods for detection of new malicious executables*. In: *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*. IEEE. 2000, pp. 38–49.

[Sch+10] Max Schuchard et al. *Losing control of the internet: using the data plane to attack the control plane*. In: *Proceedings of the 17th ACM conference on Computer and communications security*. 2010, pp. 726–728.

[Sch04] Sebastian Schmerl. *Entwurf und Entwicklung einer effizienten Analyseeinheit für Intrusion-Detection-Systeme*. PhD thesis. Diplomarbeit, Lehrstuhl Rechnernetze, BTU Cottbus, 2004.

[Sch92] I. Schaumüller-Bichl. *Sicherheitsmanagement: Risikobewältigung in informationstechnologischen Systemen*. Sicherheit in der Informations- und Kommunikationstechnik. BI-Wiss.-Verlag, 1992. ISBN: 9783411155019.

[Sch99] Bruce Schneier. *Attack trees*. In: *Dr. Dobb's journal* 24.12 (1999), pp. 21–29.

[Sek+06] Vyas Sekar et al. *LADS: Large-scale Automated DDoS Detection System.* In: *USENIX Annual Technical Conference, General Track*. 2006, pp. 171–184.

[SG06] Reza Sadoddin and Ali Ghorbani. *Alert correlation survey: framework and techniques*. In: *Proceedings of the 2006 international conference on privacy, security and trust: bridge the gap between PST technologies and business services*. 2006, pp. 1–10.

[SHM02] Stuart Staniford, James A. Hoagland, and Joseph M. McAlerney. *Practical Automated Detection of Stealthy Portscans*. In: *Journal of Computer Security* 10.1–2 (2002), pp. 105–136. ISSN: 0926-227X.

[Shy+03] Mei-ling Shyu et al. *A novel anomaly detection scheme based on principal component classifier*. In: *Proceedings of the IEEE Foundations and New Directions of Data Mining Workshop, in conjunction with the Third IEEE International Conference on Data Mining (ICDM'03)*. 2003.

[Sin92] Alistair Sinclair. *Improved Bounds for Mixing Rates of Marked Chains and Multicommodity Flow*. In: *LATIN 1992, 1st Latin American Symposium on Theoretical Informatics*. Vol. 583. 1992, pp. 474–487. ISBN: 3-540-55284-7.

[SJB04] Stuart E Schechter, Jaeyeon Jung, and Arthur W Berger. *Fast detection of scanning worm infections*. In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2004, pp. 59–81.

[SLG18] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. *Toward generating a new intrusion detection dataset and intrusion traffic characterization.* In: *ICISSP*. 2018, pp. 108–116.

[Sna+91] Steven R Snapp et al. *DIDS (distributed intrusion detection system)-motivation, architecture, and an early prototype*. In: *14th National Computer Security Conference (NCSC '91)*. 1991.

[Son+11] Jungsuk Song et al. *Statistical analysis of honeypot data and building of Kyoto 2006+ dataset for NIDS evaluation*. In: *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*. ACM. 2011, pp. 29–36.

[Sor48] Th A Sorensen. *A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on Danish commons*. In: *Biol. Skar.* 5 (1948), pp. 1–34.

[SP03] Robin Sommer and Vern Paxson. *Enhancing byte-level network intrusion detection signatures with context*. In: *Proceedings of the 10th ACM conference on Computer and communications security*. 2003, pp. 262–271.

[Spe+09] Anna Sperotto et al. *A Labeled Data Set for Flow-Based Intrusion Detection*. In: *IP Operations and Management*. 2009, pp. 39–50.

[Sta+04] Stuart Staniford et al. *The top speed of flash worms*. In: *Proceedings of the 2004 ACM workshop on Rapid malcode*. 2004, pp. 33–42.

[Sta+96] Stuart Staniford-Chen et al. *GrIDS – a graph based intrusion detection system for large networks*. In: *Proceedings of the 19th national information systems security conference*. Vol. 1. Baltimore. 1996, pp. 361–370.

[Sto+03] Ion Stoica et al. *Chord: a scalable peer-to-peer lookup protocol for internet applications*. In: *IEEE/ACM Transactions on networking* 11.1 (2003), pp. 17–32.

[Sto+09] Ben Stock et al. *Walowdac-analysis of a peer-to-peer botnet*. In: *2009 European Conference on Computer Network Defense*. IEEE. 2009, pp. 13–20.

[Str+18] Blake E Strom et al. *MITRE ATT&CK: Design and Philosophy*. Tech. rep. 2018. URL: https://www.mitre.org/sites/default/files/publications/pr-18-0944-11-mitre-attack-design-and-philosophy.pdf (visited on 08/25/2020).

[Sun+16] Xiaoyan Sun et al. *Towards probabilistic identification of zero-day attack paths*. In: *2016 IEEE Conference on Communications and Network Security (CNS)*. IEEE. 2016, pp. 64–72.

[Sze+17] Vivienne Sze et al. *Efficient processing of deep neural networks: A tutorial and survey*. In: *Proceedings of the IEEE* 105.12 (2017), pp. 2295–2329.

[Tan+14] Zhiyuan Tan et al. *Enhancing big data security with collaborative intrusion detection*. In: *IEEE cloud computing* 1.3 (2014), pp. 27–33.

[Tch+15] Taha Ait Tchakoucht et al. *Behavioral appraoch for intrusion detection*. In: *2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA)*. IEEE. 2015, pp. 1–5.

[TJ03] Marina Thottan and Chuanyi Ji. *Anomaly detection in IP networks*. In: *IEEE Transactions on signal processing* 51.8 (2003), pp. 2191–2204.

[TL01] Steven J Templeton and Karl Levitt. *A requires/provides model for computer attacks*. In: *Proceedings of the 2000 workshop on New security paradigms*. 2001, pp. 31–38.

[Val15] Harri Valpola. *From neural PCA to deep unsupervised learning*. In: *Advances in independent component analysis and learning machines*. Elsevier, 2015, pp. 143–171.

[Vas+15a] Emmanouil Vasilomanolakis et al. *SkipMon: A Locality-Aware Collaborative Intrusion Detection System*. In: *International Performance Computing and Communications Conference (IPCCC)*. 2015.

[Vas+15b] Emmanouil Vasilomanolakis et al. *Taxonomy and survey of collaborative intrusion detection*. In: *ACM Computing Surveys (CSUR)* 47.4 (2015), pp. 1–33.

[VH17] Christian Vaas and Jassim Happa. *Detecting disguised processes using application-behavior profiling*. In: *2017 IEEE International Symposium on Technologies for Homeland Security (HST)*. IEEE. 2017, pp. 1–6.

[Vin+08] Pascal Vincent et al. *Extracting and composing robust features with denoising autoencoders*. In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 1096–1103.

[VS00] Alfonso Valdes and S Skinner. *Blue sensors, sensor correlation, and alert fusion*. In: *Recent Advances in Intrusion Detection (RAID 2000), Toulouse, France* (2000).

[VS01] Alfonso Valdes and Keith Skinner. *Probabilistic alert correlation*. In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2001, pp. 54–68.

[War62] Stephen Warshall. *A theorem on boolean matrices*. In: *Journal of the ACM (JACM)* 9.1 (1962), pp. 11–12.

[Wen+15] Steffen Wendzel et al. *Pattern-based survey and categorization of network covert channel techniques*. In: *ACM Computing Surveys (CSUR)* 47.3 (2015), pp. 1–26.

[Wil+19] Florian Wilkens et al. *Towards Efficient Reconstruction of Attacker Lateral Movement*. In: *Proceedings of the 14th International Conference on Availability, Reliability and Security*. 2019, pp. 1–9.

[WLJ06] Lingyu Wang, Anyi Liu, and Sushil Jajodia. *Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts*. In: *Computer communications* 29.15 (2006), pp. 2917–2933.

[WMY18] Louis Waked, Mohammad Mannan, and Amr Youssef. *To intercept or not to intercept: Analyzing tls interception in network appliances*. In: *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. 2018, pp. 399–412.

[WN05] Amrit Williams and Mark Nicolett. *Improve IT Security with Vulnerability Management*. Tech. rep. G00127481. 2005. URL: https://www.gartner.com/en/documents/480703 (visited on 03/05/2020).

[WR03] Xinyuan Wang and Douglas S Reeves. *Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays*. In: *Proceedings of the 10th ACM conference on Computer and communications security*. 2003, pp. 20–29.

[WR10] Xinyuan Wang and Douglas Reeves. *Robust correlation of encrypted attack traffic through stepping stones by flow watermarking*. In: *IEEE Transactions on Dependable and Secure Computing* 8.3 (2010), pp. 434–449.

[WRW02] Xinyuan Wang, Douglas S Reeves, and S Felix Wu. *Inter-packet delay based correlation for tracing encrypted connections through stepping stones*. In: *European Symposium on Research in Computer Security*. Springer. 2002, pp. 244–263.

[WS02] David Wagner and Paolo Soto. *Mimicry attacks on host-based intrusion detection systems*. In: *Proceedings of the 9th ACM Conference on Computer and Communications Security*. 2002, pp. 255–264.

[WS04] Ke Wang and Salvatore J Stolfo. *Anomalous payload-based network intrusion detection*. In: *International workshop on recent advances in intrusion detection*. Springer. 2004, pp. 203–222.

[Wyk12] James Wyke. *The ZeroAccess Botnet – Mining and Fraud for Massive Financial Gain*. Tech. rep. September. Sophos, 2012.

[Xie+10] Peng Xie et al. *Using Bayesian networks for cyber security analysis*. In: *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*. IEEE. 2010, pp. 211–220.

[Yan+15] Qiben Yan et al. *Peerclean: Unveiling peer-to-peer botnets through dynamic group behavior analysis*. In: *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE. 2015, pp. 316–324.

[YBJ04] Vinod Yegneswaran, Paul Barford, and Somesh Jha. *Global Intrusion Detection in the DOMINO Overlay System*. In: *Proceedings of the Network and Distributed System Security Symposium, NDSS 2004, San Diego, California, USA*. 2004.

[Ye+01] Nong Ye et al. *Probabilistic techniques for intrusion detection based on computer audit data*. In: *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 31.4 (2001), pp. 266–274.

[YF04] Dong Yu and Deborah Frincke. *A novel framework for alert correlation and understanding*. In: *International Conference on Applied Cryptography and Network Security*. Springer. 2004, pp. 452–466.

[YF05] Dong Yu and Deborah Frincke. *Alert confidence fusion in intrusion detection systems with extended Dempster-Shafer theory*. In: *Proceedings of the 43rd annual Southeast regional conference-Volume 2*. 2005, pp. 142–147.

[ZG06] Bin Zhu and Ali A Ghorbani. *Alert correlation for extracting attack strategies*. In: *IJ Network Security* 3.3 (2006), pp. 244–258.

[ZGL18] Ru Zhang, Tao Guo, and Jianyi Liu. *An IDS Alerts Aggregation Algorithm Based on Rough Set Theory*. In: *IOP Conference Series: Materials Science and Engineering*. Vol. 322. 6. IOP Publishing. 2018, p. 062009.

[Zho+18] Donghao Zhou et al. *A survey on network data collection*. In: *Journal of Network and Computer Applications* 116 (2018), pp. 9–23.

[ZKW15] Richard Zuech, Taghi M Khoshgoftaar, and Randall Wald. *Intrusion detection and big heterogeneous data: a survey*. In: *Journal of Big Data* 2.1 (2015), p. 3.

[ZLK09] Chenfeng Vincent Zhou, Christopher Leckie, and Shanika Karunasekera. *Decentralized multi-dimensional alert correlation for collaborative intrusion detection*. In: *Journal of Network and Computer Applications* 32.5 (2009), pp. 1106–1123.

[ZLK10] Chenfeng Vincent Zhou, Christopher Leckie, and Shanika Karunasekera. *A survey of coordinated attacks and collaborative intrusion detection*. In: *Computers & Security* 29.1 (2010), pp. 124–140.

[ZP00] Yin Zhang and Vern Paxson. *Detecting stepping stones.* In: *USENIX Security Symposium*. Vol. 171. 2000, p. 184.

Acronyms

ACAG Alert Correlation with Attack Graphs

AEAD authenticated encryption with associated data

AG attack graphs

AOI Attribute-oriented Induction

APE alert partial entropy

API application programming interface

APT advanced persistent threat

BN Bayesian network

C2 command and control

CAPEC Common Attack Pattern Enumeration and Classification

CDF cumulative distribution function

CIDS collaborative intrusion detection system

CPM clique percolation method

CPT conditional probability table

CSM correlation strength matrix

DAG directed acyclic graph

DBSCAN density-based spatial clustering of applications with noise

DC domain controller

DDoS distributed denial-of-service

DFS depth-first search

DG dependency graph

DHCP Dynamic Host Configuration Protocol

DHT distributed hash table

DIDS Distributed Intrusion Detection System

DMZ demilitarized zone

DNN deep neural network

DNS Domain Name System

DRDoS distributed reflective denial-of-service

EDL Event Description Language

FPR false positive rate

FTP File Transfer Protocol

GAC graph-based alert correlation

HG hyper-alert graph

HIDS host intrusion detection system

HSG High-level Scenario Graph

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

HZCG host- & zone-communication graph

ICMP Internet Control Message Protocol

IDMEF Intrusion Detection Message Exchange Format

IDS intrusion detection system

IEP inter event pattern

IMAP Internet Message Access Protocol

IP Internet Protocol

IPS intrusion prevention system

ISP Internet service provider

kTLS in-kernel TLS

LOM local observation model

MDAC Multi-Dimensional Alert Correlation

MIME Multipurpose Internet Mail Extensions

MLP Multilayer Perceptron

NAT network address translation

NID network-user identification

NIDS network intrusion detection system

OS operating system

P2P peer-to-peer

PAC Probabilistic Alert Correlation

PDRR prevention, detection, response, and recovery

PET privacy-enhancing technology

POP3 Post Office Protocol version 3

REST representational state transfer

SCADA supervisory control and data acquisition

SDA stacked denoising auto-encoder

SIEM security information and event management

SIIT Stateless IP/ICMP Translation

SMB Server Message Block

SMTP Simple Mail Transfer Protocol

SOC security operations center

SPADE Statistical Packet Anomaly Detection Engine

SPICE Stealthy Probing and Intrusion Correlation Engine

SPOF single point of failure

SSH Secure Shell

SSL Secure Sockets Layer

STIX Structured Threat Information Expression

SVM support vector machine

TAXII Trusted Automated Exchange of Intelligence Information

TCP Transmission Control Protocol

TLS Transport Layer Security

TPR true positive rate

TSP Triad Significance Profile

TTP tactic, technique, and procedure

UDP User Datagram Protocol

UPGMA unweighted pair group method with arithmetic mean

URI Uniform Resource Identifier

URL Uniform Resource Locator

VM virtual machine

VPN virtual private network

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Dissertationsschrift selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ich bin damit einverstanden, dass meine Dissertationsschrift in den Bestand der Fachbereichsbibliothek eingestellt wird.

Hamburg, den 30.10.2020

Steffen Haas

Vorveröffentlichungen

Aus dieser Dissertation sind sechs Vorveröffentlichungen hervorgegangen. Nachfolgend sind diese mit meinem eigenen Anteil bei Konzeption, Durchführung und Berichtsabfassung im Einzelnen dargelegt.

GAC: graph-based alert correlation for the detection of distributed multi-step attacks

Referenz zum Konferenzpapier [HF18] auf der ACM SAC:

> Steffen Haas and Mathias Fischer. *GAC: graph-based alert correlation for the detection of distributed multi-step attacks*. In: *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. 2018, pp. 979–988

Dieser Beitrag wurde von mir als Erstautor weitestgehend eigenständig konzeptioniert, durchgeführt und verfasst. Der Promotionsbetreuer und Mitautor Prof. Fischer hat an der Entwicklung und Ausarbeitung der Idee, sowie der schriftlichen Abfassung unterstützend mitgewirkt.

On the Robustness of Random Walk Algorithms for the Detection of Unstructured P2P Botnets

Referenz zum Konferenzpapier [Muh+18] auf der SIG SIDAR IMF:

> Dominik Muhs et al. *On the Robustness of Random Walk Algorithms for the Detection of Unstructured P2P Botnets*. In: *2018 11th International Conference on IT Security Incident Management & IT Forensics (IMF)*. IEEE. 2018, pp. 3–14

Die technische Umsetzung dieses Beitrags sowie dessen schriftliche Abfassung wurde weitestgehend von dem studentischen Erstautor Herrn Muhs durchgeführt. Mein Anteil an diesem Beitrag als Betreuer des Studenten liegt auf dem Ansatz der Arbeit. Darüber hinaus stellt insbesondere das Konzept der Evaluation und die Interpretation der Ergebnisse meinen Anteil dar. Der Promotionsbetreuer und Mitautor Prof. Fischer hat durchgehend an der Konzeption, Durchführung und Berichtsabfassung unterstützend mitgewirkt.

On the alert correlation process for the detection of multi-step attacks and a graph-based realization

Referenz zum Zeitschriftenartikel [HF19] im ACM ACR:

> Steffen Haas and Mathias Fischer. *On the alert correlation process for the detection of multi-step attacks and a graph-based realization*. In: *ACM SIGAPP Applied Computing Review* 19.1 (2019), pp. 5–19

Dieser Beitrag erweitert das Konferenzpapier [HF18] um ein allgemeines Modell sowie weitere Experimente. Diese Erweiterungen sind ausschließlich auf mich als Erstautor zurückzuführen.

Efficient Attack Correlation and Identification of Attack Scenarios based on Network-Motifs
Referenz zum Konferenzpapier [HWF19] auf der IEEE IPCCC:

> Steffen Haas, Florian Wilkens, and Mathias Fischer. *Efficient Attack Correlation and Identification of Attack Scenarios based on Network-Motifs*. In: *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*. IEEE. 2019, pp. 1–11

Dieser Beitrag wurde von mir als Erstautor weitestgehend eigenständig konzeptioniert, durchgeführt und verfasst. Der Promotionsbetreuer und Mitautor Prof. Fischer hat an der Entwicklung und Ausarbeitung der Idee, sowie der schriftlichen Abfassung unterstützend mitgewirkt.

zeek-osquery: Host-Network Correlation for Advanced Monitoring and Intrusion Detection
Referenz zum Konferenzpapier [HSF20] auf der IFIP SEC:

> Steffen Haas, Robin Sommer, and Mathias Fischer. *zeek-osquery: Host-Network Correlation for Advanced Monitoring and Intrusion Detection*. In: *ICT Systems Security and Privacy Protection*. Springer, 2020

Dieser Beitrag wurde von mir als Erstautor weitestgehend eigenständig konzeptioniert, durchgeführt und verfasst. Die Idee basiert auf einer Vorarbeit des Mitautors Herrn Sommer. Darüber hinaus, hat dieser vor allem die Entwicklung des Open Source Prototypens unterstützt, sowie Anwendungsszenarien mitentwickelt. Der Promotionsbetreuer und Mitautor Prof. Fischer hat an der Entwicklung und Ausarbeitung der Idee, dem Konzept einer realistischen Evaluation, sowie der Abfassung unterstützend mitgewirkt.

Scan Correlation – Revealing distributed scan campaigns
Referenz zum Workshoppapier [HWF20] auf der IFIP NOMS:

> Steffen Haas, Florian Wilkens, and Mathias Fischer. *Scan Correlation – Revealing distributed scan campaigns*. In: *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE. 2020

Dieser Beitrag basiert auf der von mir betreuten Bachelorarbeit [Reg19], in der der Student meinen Ansatz und mein Evaluationskonzept technisch umgesetzt hat. Die Vorveröffentlichung wurde von mir als Erstautor verfasst. Der Promotionsbetreuer und Mitautor Prof. Fischer hat vor allem an der initialen Idee und der schriftlichen Abfassung unterstützend mitgewirkt.