Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

UNIVERSITÄT HAMBURG

DISSERTATION

Methods for Enhanced Security Monitoring and APT Detection in Enterprise Networks

---

vorgelegt von

Florian WILKENS

Dissertation zur Erlangung des Titels
Dr. rer. nat.

an der

Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik
Arbeitsgruppe Rechnernetze

eingereicht am 15.11.2022

Reviewers

Prof. Dr. Mathias Fischer
Prof. Dr. Christian Dörr

Date of the oral defense

28.04.2023

Unique Resource Name (URN)

urn:nbn:de:gbv:18-ediss-110970

Checksum of the submitted version (SHA-256)

c055dea8283aaf57d6e4fe2759648ba38bb4e369d8f2c48ebbfeb8902aae2c99

"The only truly secure system is one that is powered off, cast in a block of concrete and sealed in a lead-lined room with armed guards—and even then I have my doubts."

Eugene Spafford

Abstract

Methods for Enhanced Security Monitoring and APT Detection in Enterprise Networks

by Florian Wilkens

Nowadays, pervasive attacks on IT systems have become the new normal ranging from simple automated attacks to sophisticated and stealthy nation-state attacks, so-called advanced persistent threats (APTs). To protect against these attacks, organizations utilize security operations centers (SOCs) with expert analysts who investigate the vast stream of alerts generated by security infrastructure. Detection of APTs is limited by two central factors: missed event- and alert-data due to insufficient security monitoring and analysts not investigating key alerts due to alert fatigue from the high overall alert volume. Thus, this thesis presents five contributions to (*i*) *enhance security monitoring* for better visibility on APT activity and (*ii*) support SOC analysts via *novel mechanisms for APT detection* to cope with the high volume of alerts.

The first two contributions towards security monitoring increase both quality and quantity of internal and external visibility. We introduce a concept for passive and transparent TLS decryption via key sharing of cooperative endpoints. In contrast to TLS interception via proxy servers, our approach is computationally more efficient and prevents a major attack vector, as absolute trust in a single proxy is no longer required. Our results indicate that transfer latency of key material shared by the endpoints can be addressed by a small traffic buffer of about 40ms. In addition, we present an approach to characterize brute-force attacker behavior via a collection of established features and two novel metrics that we leverage to cluster and prioritize attackers. In our evaluation on a real-world dataset of login attempts, we highlight how our metrics helps to establish indicators for likely collusion between unrelated IP addresses and potential APT reconnaissance activity of unclustered login attempts.

The remaining three contributions of this thesis support SOC analysts via novel detection algorithms for APT stages and campaigns. We present an abstract algorithm to reconstruct lateral movement activity based on security relevant host information and incomplete alert sets as well as two concrete implementations. Our evaluation indicates good detection performance for the variant based on k-shortest paths with up to 90% true positive rate for insider attackers that deviate 80% from the expected path. Furthermore, we describe a concept to explain classification decisions of graph-based APT detection approaches that rely on anomaly detection. We achieve this by systematically altering the input graphs and observing the changing anomaly scores. The result is added context that is essential for quick incident response. Last, we introduce an approach to reconstruct entire APT campaigns based on a novel Kill Chain State Machine (KCSM) that is derived from established kill chain models. The algorithm aggregates and links alerts along the state machine to obtain a compact visual representation of the APT campaign that helps analysts to quickly assess it. Our results show that our approach massively reduces the alert volume down to 0.3–3.5% of the original alert set, while retaining the contextual information of the APT campaign. Concluding, our five proposed approaches can be used individually or in conjunction to improve the APT detection capabilities of a SOC for enterprise networks: enhanced security monitoring helps to obtain indicators of APT activity, while our mechanisms for detection support SOC analysts in coping with alert fatigue.

Zusammenfassung

Anriffe auf IT-Systeme sind heute zur Normalität geworden. Diese reichen von simplen automatisierten Attacken bis zu komplexen nation-state Angriffen, den sogenannten advanced persistent threats (APTs). Organisationen nutzen security operations centers (SOCs) mit Analysten, um sich gegen diese Angriffe zu schützen. Die Analysten untersuchen den Strom an generierten Alarmen der Sicherheitsinfrastruktur. APT Detektion ist von zwei zentralen Faktoren beschränkt: fehlenden Ereignis- und Alarmdaten aufgrund von unzureichendem Sicherheits-Monitoring und Analysten, die von der hohen Alarmanzahl überfordert sind und daher wichtige Alarme übersehen. Diese Dissertation präsentiert fünf Beiträge, die (*i*) *Sicherheits-Monitoring verbessern*, um erhöhte Sichtbarkeit auf APT Aktivität zu erhalten und (*ii*) Analysten mit *neuen Mechanismen für APT Detektion* unterstützen, das hohe Alarmvolumen zu bewältigen.

Die ersten beiden Beiträge im Bereich Sicherheits-Monitoring verbessern Qualität und Quantität der intern und extern erreichten Sichtbarkeit. Zuerst wird ein Konzept für passive und transparente Entschlüsselung von TLS-Verbindungen mittels kooperativen Hosts vorgestellt. Im Gegensatz zum Aufbrechen von TLS mittels Proxyservern, ist dieser Ansatz weniger rechenintensiv und minimiert eine zentrale Angriffsmöglichkeit, da absolutes Vertrauen in einen Proxyserver nicht mehr erforderlich ist. Die Ergebnisse zeigen, dass die Übertragungslatenz von Schlüsselmaterial mithilfe eines Netzwerkpuffers von 40ms adressiert werden kann. Außerdem wird ein Ansatz zur Charakterisierung des Verhaltens von brute-force Angreifern vorgestellt, der auf einem Set an etablierten und zwei neuen Metriken basiert. Diese werden verwendet, um Angreifer zu gruppieren und zu priorisieren. Die Evaluation auf einem Echtweltdatensatz zeigt, wie die neuen Metriken helfen, mögliche Kollusion zwischen unverbundenen IP Adressen und potentielle APT Aktivität, in Form von Reconnaissance, aufzudecken.

Die verbleibenden drei Beiträge dieser Dissertation unterstützen SOC Analysten mittels neuartiger Ansätze zur Detektion von APT Phasen und Kampagnen. Es wird ein Algorithmus sowie zwei Implementationen zur Rekonstruktion von Lateral Movement vorgestellt, die sicherheitsrelevante Host-Informationen und unvollständige Alarmsets kombinieren. Die Evaluation der Variante basierend auf k-kürzesten Pfaden zeigt eine Sensitivität von 90% für die Detektion von Insider-Angreifern, die 80% vom erwarteten Pfad abweichen. Außerdem wird ein Konzept zur Erklärung von Klassifikationsentscheidungen von graphbasierten APT Detektionsansätzen beschrieben, die auf Anomalieerkennung basieren. Dies wird erreicht, indem Eingabegraphen systematisch angepasst und die sich ändernden Anomaliewerte beobachtet werden. Der zusätzliche Kontext hilft bei einer schnellen Vorfallsbehandlung. Zuletzt wird ein Ansatz zur Rekonstruktion von APT Kampagnen vorgestellt, der auf einer neuartigen Kill Chain State Machine (KCSM) basiert, die von etablierten Kill Chain Modellen abgeleitet ist. Der Algorithmus aggregiert und verknüpft Alarme entlang des Automaten, um eine kompakte Repräsentation der Kampagne zu erhalten, die Analysten in deren Beurteilung unterstützt. Die Ergebnisse zeigen, dass das Alarmvolumen massiv auf 0.3–3.5% reduziert wird, während Kontextinformationen über die APT Kampagne beibehalten werden. Zusammenfassend können die fünf vorgestellen Ansätze individuell oder zusammen verwendet werden, um die APT Detektionsfähigkeiten eines SOC für Unternehmensnetzwerke zu stärken: verbessertes Sicherheits-Monitoring hilft Indikatoren für APT Aktivität zu erhalten, während die Mechanismen zur Detektion Analysten unterstützen, das hohe Alarmvolumen zu bewältigen.

Acknowledgements

This thesis is the result of not only the time and effort invested by me, but also the understanding and support of all those around me. This achievement would have been impossible without you.

First and foremost, I want to thank my adviser Mathias Fischer, who roped me into academia and offered me the chance to work on interesting problems with the convenient side effect to potentially earn a title at the end. Your encouragement and valuable discussions shaped and improved this thesis tremendously. I also want to thank my collaborators, namely Peter Kling and Dominik Kaaser for ensuring my first paper stood on a solid formal foundation, Matthias Vallentin for your valuable insights from "the real world", and Johanna Amann for graciously offering both your technical and academic expertise for everything related to Zeek. Similarly, I owe a sincere thanks to the hard work of all my students, in particular Felix Ortmann, Felix Welter, Jona Laudan, Henning Schütt, and Julian Frangopoulos. Your ideas and our discussions were essential for this thesis.

I am genuinely grateful to all my colleagues from the working group "Security and Privacy" who made my time as a PhD candidate enjoyable and fun. The thoughtful discussions and overall atmosphere helped to make this possible. I fondly remember our everchanging talks over lunch, shared retreats, and spontaneous hall conversations. Thank you Tatjana (Wingarz), Monina (Schwarz), Nurefsan (Sertbas-Bülbül), Christian (Burkert), Tobias (Müller), Jens (Wettlaufer), and Matthias (Marx). A special shoutout to our "Viererbüro" which provided just the right environment to make significant progress in a publication or spend a relaxing day in the office couch after an exhausting deadline. Thank you, Steffen (Haas) for being an overall awesome friend and for signaling the rest of us to get back to work by putting your headphones on when the office became too noisy. Thank you, David (Jost) for your tireless efforts to help me fix Latex, Python, or any other technical problem I encountered during this thesis. Thank you, Doğanalp (Ergenç) for motivating the whole office to squeeze in some exercise into the work day and for your unrivaled talent to bring a smile to everyone's face. Last but not least, Tom (Petersen), some day I will beat you in chess without you going easy on me. I am happy to have shared this journey with you and thankful to be able to call all of you my friends.

My family also played a large role in the success of this thesis. My Mom Sabine, who immediately encouraged me to start the PhD when I still had doubts. My Dad Sven and his wife Kristin, who never failed to ask about my progress and dutifully listened to my increasingly complex explanations. My brother Jannik, who dealt with my varying time budget perfectly and always supported me in his own special way. I know, I can always rely on all of you.

Finally, I want to thank my long-term partner Nadège for your unwavering support and encouragement throughout the whole time I worked on this thesis. Out of all people around me, you probably had to endure the most of my rants and doubts, yet you never failed to find the right words to keep me motivated and confident. Thank You for sticking with me through this intensive time!

Contents

Dedicated to my Mom

1 Introduction

In our digitized world, businesses, governments, and other large organizations heavily rely on their IT infrastructure to operate and create value for their customers. Naturally, this is not news to cybercriminals who seek to monetize these dependencies by stealing sensitive data, deploying ransomware, or finding other creative ways to extract money from their malicious activity. This results in a multitude of cyber attacks on every system that is available to the open Internet ranging from generic password guessing and simple denial of service (DoS) attacks to sophisticated nation-state level attacks specifically tailored to their victims. These so-called advanced persistent threats (APTs) pose a severe risk especially to governments, critical infrastructure and other large organizations and companies as they often go unnoticed for weeks to months and infiltrate large amounts of hosts across many subnets. While the concrete goals differ greatly between attack campaigns, APTs are often politically motivated and aim to maximize damage to their targets. This is in stark contrast to the usual goal of cybercriminals to maximize monetary profits. Examples like Stuxnet in 2005–2010 [Zet14], the attacks on the German Bundestag in 2015 [Beu+17] and Democratic National Committee in 2016 [CNN16], and most recently the SolarWinds supply chain attack in 2020 [CJ21] highlight the threat and various types of damages (political, financial, reputational, and legal) APTs can inflict on their victims.

1.1 Problem Statement and Research Questions

Detection and mitigation of APT campaigns is an open challenged tackled by both industry and academia. The process is limited by two major factors: (*i*) Due to their low-and-slow nature, APTs leave only minimal traces on compromised hosts or in the network resulting in very few alerts. Attackers often completely *evade security monitoring* such that campaigns are detected months to years later after significant damage has been done. (*ii*) If alerts (or events) are generated for APT activity, they are greatly outnumbered by the flood of unrelated and false-positive alerts that security analysts need to investigate. In the worst case, this *alert fatigue* causes them to miss the few key alerts that may help to reveal the larger APT campaign.

To defend against these attacks and protect their assets, large organizations deploy comprehensive *security monitoring* including network monitoring systems (NMSs) and intrusion detection systems (IDSs) to obtain as much visibility into both internal and external activity as possible. Both aspects are important for APT detection, these complex attack rely on significant adversarial infrastructure that interacts with the compromised hosts in the target network. However, existing systems and approaches are not sufficient in matching the stealthiness of APT attacks and fail to detect key activity like encrypted command & control (C2/C&C) communication or single login attempts that is performed as part of target reconnaissance. The other important aspect of the security infrastructure is the security operations center (SOC), a central entity in the organization where events and alerts obtained via security monitoring are collected

and further investigated. These *security analytics* approaches include fully-automated processing concepts like alert correlation, human expert analysis that is supported by semi-automated preprocessing and even proactive threat hunting for unknown threats. While there are some approaches for mostly automated APT detection, they are either (*i*) employ complex models that fail to provide the additional attack context required for incident response or (*ii*) require highly detailed system information that is practically infeasible to capture reliably in large enterprise networks.

Thus, contemporary research in the area of APT detection usually aims to support the human SOC analysts in their manual investigative work. While the overall arms-race between (APT) attackers and defenders is unlikely to end any time soon, sophisticated automated analysis and preprocessing in combination with manual human-in-the-loop expert investigation remains our best shot at detecting and mitigating these attacks. Approaches that increase the effectiveness of this process, can help to reduce the risk of complex cyber attacks. In summary, the overall focus of this thesis lies on the detection of APT campaigns by enhancing both security monitoring and security analytics to effectively support SOC analysts. More concretely, the following research questions are answered in this thesis:

- **RQ1**: *How can NMSs regain visibility into TLS-encrypted communication without actively intercepting the connections on a central entity and breaking end-to-end guarantees?*
- **RQ2**: *How and to what extend can brute-force attempts on externally-accessible services be categorized and screened for potential APT reconnaissance activity?*
- **RQ3**: *How can APT mitigation be supported by combining both alerts and security-relevant host information, e. g., to estimate impacts of the attack?*
- **RQ4**: *Which techniques from explainable artificial intelligence (XAI) can be leveraged to add attack context to classification decisions made by anomaly detection-based APT detection approaches?*
- **RQ5**: *How can established kill chain-based APT models be leveraged for campaign detection with only minimal assumptions about the lower-level alert set?*
- **RQ6**: *Which level of volume reduction can be achieved during APT campaign reconstruction to lessen the impact of alert fatigue on SOC analysts?*

The next section briefly summarizes the contributions made in this thesis to answer these questions.

1.2 Contributions

This section briefly summarizes the main contributions of this thesis (as also listed in Appendix A.1) and explains how they relate to the research questions. The Appendix also lists some additional publications that the author of this thesis contributed to (in Appendix A.2) as well as published datasets (in Appendix A.3) and supervised theses (in Appendix A.4).

Passive, Transparent, and Selective TLS Decryption for Network Security Monitoring

Encrypted traffic is becoming increasingly popular both for internal communication in networks as well as on the public Internet. While this has several benefits for overall privacy of users and organizations alike, the ubiquitous encryption with protocols like TLS poses problems for NMSs and IDSs especially in enterprise networks. These tools lose visibility into the (now encrypted) application-level payloads and thus need to fall back on metadata-only analysis. Although simple checks and mitigations like IP-based detection and blocking remains possible, complex analysis tasks like signature-based malware detection or detection of data exfiltration become hard to impossible without cleartext payloads. The predominant approach to address this problem in enterprise settings is TLS interception via Man-in-the-Middle (MitM) proxy servers (sometimes also referred to as "split" TLS). The proxy can then forward cleartext payloads to the NMS or IDS. However, this poses several security and privacy issues as outlined by several researchers in [Jar12], [dCM16], and [Dur+17]. As a consequence, multiple extensions to standard TLS have been proposed that add explicit support for middleboxes including NMSs and IDSs such as Multi-Context TLS (mcTLS) [Nay+15], Middlebox TLS (mbTLS) [Nay+17], Locally Operated Cooperative Key Sharing (LOCKS) [Bie+17], and Middlebox-aware TLS (maTLS) [Lee+19]. However, at the time of writing, no proposal has managed to achieve significant adoption mainly due to problems with incremental deployment.

This thesis proposes a generic system for passive inspection of TLS-encrypted communication on trusted NMS without the need for a MitM proxy. Instead, we suggest that endpoints selectively forward established TLS key material after handshake completion to the NMS, so that it can passively decrypt and analyze cleartext payloads without actively intercepting or otherwise modifying the connection. In contrast to other state-of-the-art approaches like TLS interception via MitM or the aforementioned TLS extensions [Nay+15; Nay+17; Bie+17; Lee+19], our approach: (*i*) works without modifications of the TLS protocol, (*ii*) adds no additional latency to TLS session establishment or data transfer, (*iii*) enables only trusted observers to inspect cleartext payloads of TLS connections, (*iv*) supports policy-based, partial traffic inspection, that is enforced directly on endpoints, and (*v*) conceptually preserves the end-to-end integrity of the TLS session for non-AEAD cipher suites as the NMS only receives decryption keys but no integrity keys. To evaluate our approach, we built a prototype implementation of TLS decryption as a module for the Zeek NMS [Zee22]. We then evaluated our approach in two experiments to measure the added overhead introduced by performing TLS decryption in the NMS and the impact of key transfer latency on decryption success rate.

This work has been peer-reviewed and published at IFIP SEC 2022 [Wil+22] prior to publication of this thesis where it received both "*Best Student Paper*" and "*Best Paper*" awards. The evaluation dataset [WHF22] was also published. Additionally, an extended version of our prototype implementation of TLS decryption has been merged into Zeek mainline as is publicly available starting from Zeek version v5.0.0 [AW22].

Data-driven Characterization of Brute-force Attackers

In their day-to-day work, SOC analysts and threat hunters are confronted with a continuous stream of alerts that are generated by sensors, NMSs, and IDSs across the organization's network. Externally-accessible hosts are often among the hosts that

are referenced the most in this alert stream, as they are targeted by both automated scanners and manual attacks from the open Internet, e. g., by brute-force login attempts on services such as telnet, SSH, or RDP. While the majority of these attacks can be disregarded as regular Internet noise without meaningful threat, login attempts are also part of reconnaissance activity as the first step of APT campaigns or other complex attacks. Thus, SOC analysts need to carefully filter and prioritize login attempts based on structural and behavioral metrics like IP address and dictionary. Several of these metrics have been described by Owens and Mattews [OM08] and Abdou et al. [ABvO16].

This thesis proposes two novel concepts to capture detailed attacker behavior from brute-force login attempts: (*i*) *brute-force sessions* extend the existing timing-based metrics and group multiple login attempts of the same IP address that are observed in a defined timespan as a session. We then derive two metrics from the sequence of sessions, namely Time-between-Logins (TbL) (per session) and Time-between-Sessions (TbS) (across sessions). This provides fine-grained insights about the timing behavior of returning attackers that were not previously considered. (*ii*) *dictionary entropy* describes the information-theoretic complexity of the sequence of credentials used by the attacker. While the attacker dictionary (also commonly referred to as "password list") itself has been analyzed by prior work, the dictionary entropy offers novel insights into the changes in credential complexity as it evolves over time and additional login attempts. In our evaluation, we show how the overall set of features and metrics (both established and newly proposed) can be used in combination to (*i*) cluster IP addresses with similar behavior and (*ii*) prioritize single IP addresses or clusters based on the expected threat level. For this, we captured a dataset of real-world brute-force login attempts on a publicly-accessible instance of our Honeygrove [Hon+22] honeypot. This dataset contains about 2 000 000 malicious login attempts captured in a three-month period.

This work has been peer-reviewed and published at IEEE CNS 2020 [WF20b] prior to publication of this thesis. Additionally, the accompanying BLF2020 dataset [WF20a] has been published to foster further research in this area.

Efficient Reconstruction of Attacker Lateral Movement

APT campaigns generally consist of a sequence of attack stages (that we describe in more detail in Section 2.2) from reconnaissance and malware delivery up to objective actions such as data exfiltration. Among these stages, lateral movement, i. e., the compromise of new hosts in a previously entered network zone or new network zone (depending on the definition used), is often highlighted as a key stage, as the attacker solidifies their foothold in the network and obtains access to new hosts. Once a campaign is detected, SOC analysts are tasked with identifying and cleaning the hosts that were compromised during lateral movement. Based on an incomplete set of alerts or indicators of compromise (IoCs), they aim to prioritize the set of potential hosts for manual time-intensive investigation. This process is time-critical for two reasons: (*i*) any hosts that are overlooked or investigated too late might be leveraged by the attacker to re-enter and persist in the network and (*ii*) the set of compromised hosts is essential to estimate the potential damage the APT campaign has caused, e. g., by compromising the central database server and exfiltrating sensitive customer data.

This thesis proposes a novel approach to reconstruct attacker lateral movement based on a graph-based formal model consisting of a host model and three derived attacker

classes ranging from basic attackers without any knowledge about the network's inner workings to a full-blown insider attacker with complete knowledge about all hosts in the network. We then present an abstract algorithm that iteratively approximates the unknown set of compromised hosts from the input alert set. The resulting ordered set of hosts can then be leveraged by SOC analysts during their investigations. We also show two concrete variants of the algorithm based on k-shortest path and biased random walks. For our evaluation, we then generate network topologies according to our graph-based network model as well as different attacks according to our attacker model. As a real-world attacker might not exactly fall into one of our three attacker classes, we also investigate how deviation from the idealized models impact detection performance. This work has been peer-reviewed and published at ARES 2019 [Wil+19a] prior to publication of this thesis.

Explainability for APT Detection on System Provenance Graphs

Anomaly detection on (whole-)system provenance graphs is an active area of research that regularly achieves high detection performance. Approaches like UNICORN [Han+20] and Pagoda [Xie+20] consist of carefully trained normality models that capture fine-grained host behavior and can thus detect even slight deviations as anomalies. However, the complexity of the underlying model is also a large disadvantage as it hinders explainability of classification decisions. This leaves SOC analysts with an opaque alert that flags a certain host or subnet as compromised without additional details, thus slowing mitigation and other incident response measures. In recent years, the area of XAI has become popular as researchers in academia and industry recognized the importance of reasoning for artificial intelligence (AI)-based approaches across domains. Approaches like LIME [RSG16] and SHAP [LL17] aim to explain single classification decisions or even entire models. However, they generally focus on tabular data as the predominant format in data science.

This thesis proposes a general approach for added explainability for graph-based anomaly detection approaches. More specifically, we apply it in the context of anomaly detection-based APT detection on system provenance graphs. The concept is similar to LIME [RSG16] and leverages a variation of permutation importance by systematically modifying input graphs and observing the changing anomaly score they result in when fed to the model to explain. This has the added benefit that our approach treats the anomaly detection approach as a black box and thus is applicable to any whole-graph classification problem. At the time of writing, this work is in the process of peer-review and submitted to IEEE ICC 2023 [WWF23].

Multi-stage Attack Detection via Kill Chain State Machines (KCSM)

The most complex task SOC analysts and threat hunters are faced with today is the detection of complete APT campaigns as they emerge in the network. While some of the previously presented contributions of this thesis help to prioritize (and potentially filter) the continuous alert stream, the remaining alerts still have to be manually analyzed. Due to the complexity of APT attacks in general and differences between specific campaigns, it is unlikely this task can be completely automated. However, several approaches have been proposed to reconstruct potential APT activity along the established kill chain models [HCA11; Pol21] to support the triage and analysis processes. Approaches like HOLMES [Mil+19b], NoDoze [Has+19], and POIROT [Mil+19a] (among others) generate graphical representations of the complex interlinked information present in the alert and event data to enable analysts to quickly

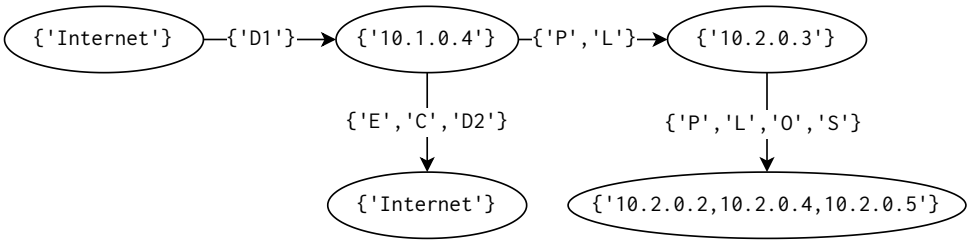


FIGURE 1.1: Example APT scenario graph

identify potential APT campaigns. However, most of these approaches rely on detailed system provenance data, i. e., kernel-level logs about system activity, that is (*i*) high volume, (*ii*) hard to capture reliably, and (*iii*) imposes a non-negligible performance impact on the monitored hosts.

This thesis proposes a novel approach to contextualize and enrich existing alerts based on a state machine that is derived from the comprehensive unified kill chain (UKC) model [Pol21]. We leverage this so-called Kill Chain State Machine (KCSM) to link potential APT activity and to synthesize a comprehensive visual representation that we call APT scenario graphs. An example of such a graph is shown in Figure 1.1 containing sets of IP addresses as nodes and potential APT stage transitions as edges. This enables SOC analysts to quickly grasp the affected hosts and the complexity of the detected scenario. Compared to prior work, our approach does not rely on system provenance but can in fact consume any network-based alert (without attached stage labels) or pre-tagged alerts that may be generated by an existing security information and event management (SIEM) or any other established approach for single-stage APT detection. For untagged alerts, our approach then derives potential APT stages based on the network direction of the alert. We then combine the input alert set with pre-configured topology information to trace and reconstruct potential APT activity in the network. Additionally, our approach offers scenario prioritization based on a configurable scoring system to further support SOC analysts throughout the remaining triage and analysis phase. Large parts of this work have been peer-reviewed and published at CYSARM 2021 [Wil+21] prior to publication of this thesis.

Chapter	Section	RQs	Contribution
Chapter 1: Introduction			
Chapter 2: Background			
Chapter 3: Requirements and State of the Art			
Chapter 4	4.1	RQ1	IFIP SEC 2022 [Wil+22; WHF22]
	4.2	RQ2	IEEE CNS 2020 [WF20b; WF20a]
Chapter 5	5.1	RQ3	ARES 2019 [Wil+19a]
	5.2	RQ4	*Submitted to:* IEEE ICC 2023 [WWF23]
	5.3	RQ5,RQ6	CYSARM 2021 [Wil+21]
Chapter 6: Conclusion			

TABLE 1.1: Thesis Overview & Contributions

1.3 Outline

Table 1.1 gives an overview about this thesis and its contributions. This thesis is structured as follows: **Chapter 2** provides required background for this thesis on the term advanced persistent threat (APT), models for APTs, and TLS fundamentals about handshakes.

Chapter 3 derives requirements for comprehensive APT detection in large-scale networks and proposes a taxonomy for relevant literature in this area. Additionally, these works are described in a detailed literature survey and compared to the formulated requirements.

The two approaches presented in **Chapter 4** improve the visibility obtained by security monitoring. The additional knowledge obtained this way, helps SOC analysts to accurately judge the current threat landscape and activity in the internal network and lays the foundation for additional higher-level correlation and detection approaches.

Chapter 5 describes three approaches that aim to improve the overall detection of APTs. Due to the complex nature of this problem, three different areas are addressed: (*i*) a stage-specific detection approach for lateral movement activity, (*ii*) a campaign-based detection algorithm aiming to reveal complex stage chains, and (*iii*) an auxiliary approach to improve explainability for AI-based approaches for APT detection.

This thesis concludes in **Chapter 6** by summarizing the contributions made and providing an outlook for future work in the areas of securitymonitoring and APT detection.

2 Background

This chapter provides some background information required for this thesis. First, in Section 2.1, the term advanced persistent threat (APT) is defined in the context of this thesis based on existing definitions from literature. Next, in Section 2.2 several established models that capture APT behavior are discussed. Lastly, in Section 2.3, we briefly summarize information about handshakes and key derivation in the TLS protocol.

2.1 APT Definition

APTs and their detection are central to this thesis and as such we need to define the term precisely before we can move on to the main contributions. The term APT was coined in 2007 by United States Air Force Colonel Greg Rattrey [Bej10; Hol13; Bej20] likely to describe classified threat actors with non-military personnel. While the exact details are unknown, the term was quickly picked up by security vendors especially after Mandiant (later acquired by FireEye) released their report on Chinese espionage activity by PLA Unit 61398 nicknamed "APT1". The term has since gained traction in the industry to describe both the overall threat landscape of nation-state level attack campaigns as well as specific threat actors (and groups) such as the aforementioned APT1[1], APT28 (presumably a Russian military unit)[2], and APT41 (another presumably Chinese state-sponsored threat group)[3].

Definitions of what exactly represents an APT also differ between security vendors and organizations. In their initial M-Trends report in 2010 [Man10], Mandiant characterizes APTs as highly skilled and professional attackers that target primarily government and defense organizations as well as large private companies and organizations. Primary goals are not financially motivated but rather focus on obtaining an informational advantage by exfiltrating sensitive information from their targets. The report also highlights stealthy persistence as a differentiating factor to traditional attacks as APTs adversaries aim to return later and steal more data. Furthermore, it is assumed that APTs are state-sponsored as this explains the scale and complexity of the attacks. While Mandiant attributes nearly all observed APT activity to China, it is important to note that the report only contains attacks up to the publication data in 2010 and that dependable attribution for cyber attacks is difficult in general.

Also in 2010, TaoSecurity's Robert Bejtlich describes four key categories that all APT objectives fall into: (*i*) *economic* objectives that include theft of proprietary intellectual property of data either for direct financial gain or competitive advantage, (*ii*) *political* objectives (e. g., suppressing "conflicting" ideologies in domestic settings or affecting opinions in foreign discussions), (*iii*) *technological* objectives that further the group's

[1] see: https://attack.mitre.org/groups/G0006/
[2] see: https://attack.mitre.org/groups/G0007/
[3] see: https://attack.mitre.org/groups/G0096/

other interest, e. g.,, by stealing source code or studying defenses to improve later attacks, and (*iv*) *military* objectives that aim to gain advantages for ongoing or potential future combat situations [Bej10]. Especially the last two categories highlight the difference between APTs and other complex attack campaigns that are usually focused on direct financial gain.

A common definition for advanced persistent threat (APT), that is often referenced in both academia and industry, was published by US National Institute of Standards and Technology (NIST) in 2011:

> *"An advanced persistent threat is an adversary that possesses sophisticated levels of expertise and significant resources which allow it to create opportunities to achieve its objectives by using multiple attack vectors (e.g., cyber, physical, and deception). These objectives typically include establishing/extending footholds within the information technology infrastructure of the targeted organizations for purposes of exfiltrating information, undermining or impeding critical aspects of a mission, program, or organization; or positioning itself to carry out these objectives in the future. The advanced persistent threat: (i) pursues its objectives repeatedly over an extended period of time; (ii) adapts to defenders' efforts to resist it; and (iii) is determined to maintain the level of interaction needed to execute its objectives."* — NIST Special Publication 800–39 [Ros11]

This definition highlights the complexity of APTs in regards to their attack vectors that are usually not limited to just information systems and their extended persistence over large periods of time. However, the description of objectives is rather limited only mentioning data exfiltration or disruption of core services. While these are relatively concrete actions an APT might take, the larger motivations behind the attack are not covered by NIST's definition.

We derive a definition of the term "advanced persistent threat (APT)" in the context of this thesis based on Bejtlich's descriptions of objectives [Bej10] and NIST's overall definition [Ros11]:

Advanced refers to the capabilities of the attacker both in terms of resources and knowledge. While the term *advanced* alone might describe various complex attacks, APTs nearly always describe nation-state level attacks which is also represented in the campaign goals.

Persistent refers to the extensive time which APT campaigns span and remain in the target network. Compared to regular attacks, APT attackers move stealthily and try to stay undetected. Post-mortem analysis of these campaigns often shows activity spanning multiple months or years.

Threat refers to the vast damages that APT campaigns cause in the financial, legal or reputational sense. APTs often aim to manipulate core services and/or exfiltrate sensitive data of the target organization, the consequences are severe. A significant difference to regular cyber attacks is the focus on non-financial objectives such as political agendas as shown in the attack on the German Bundestag [Beu+17] and the Democratic National Committee in the US [CNN16].

This definition supports the research questions of this thesis: (*i*) the *advanced* nature complicates the reliable detection of all attack steps of APT campaigns, (*ii*) the *persistent* and stealthy nature of the long-spanning campaigns further detriments the correlation and contextualization of detected events, and (*iii*) the high *threat* of APTs

motivates our overall research into this direction to achieve more accurate and timely detection.

2.2 APT Models

The modeling of APT campaigns and attacks has been the target of both academic and industrial research over the years. Several models have been proposed to describe multi-stage attacks in general and APTs more specifically, most of them based on the notion of a sequence of attack stages. This section describes the most important works in this field.

2.2.1 Intrusion Kill Chain (IKC)

The most commonly known model for APTs is the *intrusion kill chain* (*IKC*) introduced by Lockheed Martin in 2011 [HCA11]. Heavily influenced by the background of the company, the term is derived from kill chains that are used to model military attacks in the physical world. The IKC consists of seven distinct stages that describe attacker actions in APT scenarios. It describes seven consecutive stages that are executed sequentially in a complex cyber attack such as an APT campaign. *Note*: While the term IKC was used in the original academic publication, Lockheed Martin uses the term *Cyber Kill Chain®* to refer to their model in marketing material and usage in academia is mixed.

1. Reconnaissance: This stage describes the "research, identification and selection of targets" [HCA11] that APT adversaries perform before any direct attack takes place on the target organization or network. Nowadays, this often includes open-source intelligence (OSINT) from public sources such as social media or promotional material.
2. Weaponization: In this stage, the adversary *weaponizes* an exploit by combining it with malware that later establishes remote control. The exploit may be either technical in one of the organizations publicly accessible services or in internally used software (such as office suites or document viewers) or organizational, i. e., weak links in the personnel or processes.
3. Delivery: This stage describes the process of transmitting the weaponized malware to the target network or organization. Depending on the exploit that was used, the delivery vector can be an attachment of a malicious email, a compromised website that an employee is lured to, a direct exploit of a public service or even a carefully placed USB drive with that contains the malware.
4. Exploitation: In this stage, execution of the malware is achieved by either actively exploiting a component inside the target network or other means such as automatic execution from removable media or tricking the victim into macro execution in office documents.
5. Installation: This stage describes the process of the malware persisting on the initially infected device.
6. Command & control (C2/C&C): In this stage, the infected device contacts a adversary-controlled server outside of the target environment to signal successful infection and establish remote access. Once the channel to the C2/C&C

server is established, the attacker has complete control over the infected device (within the permission set of the executed malware).

7. Actions on Objectives: This rather abstract stage describes all actions the APT adversary aims to achieve in the target network such as data exfiltration including collection and encryption thereof or manipulation of internal services. The IKC also explicitly places lateral movement, i. e., compromise of additional systems in the target network, in this stage.

After the IKC stages are introduced, the publication provides examples how it can be used to better understand APT intrusions by either backtracking through the stages once a later stage has been detected or synthesizing the following stages from an incident to anticipate the next steps in the campaign. Furthermore, the authors explain how the analysis of multiple intrusions over long timespans can leverage the IKC to find overlap and thus correlate the intrusions.

The IKC provides a solid model to describe large parts of an APT attack campaign. However, researchers and other security experts have since identified several weaknesses with it [Eng14]:

- **Strictness**: The IKC explicitly assumes a sequential model of all six stages and states that an attack campaign can only proceed from one stage to the next once its objectives are met. This implies that a valid defense strategy can be centered around stopping a single stage which thus prevents the overall attack from proceeding. While this is certainly true for some stages, later reports have shown that APT campaigns can also skip certain stages or execute them in another order. Examples for this would be insider attacks that may lack any execution or remote control as the adversary uses regular tooling from withing the organization or an APT campaign that first establishes remote control through C2/C&C before persisting.
- **Coarseness/Missing stages**: As mentioned before, the last stage of the IKC, *Action on Objectives*, is highly coarse and captures a wide range of attacker behavior after the initial compromise such as data collection, exfiltration and lateral movement. While the single stage allows the model to stay generic and map to many APT campaigns, the lack of details also makes this stage hard to detect and mitigate.
- **Lack of domain-specifics**: The IKC was proposed as a general model for APT campaigns and thus does not tailor to specialized domains of either APT campaigns or application domains. While not a weakness of the model in itself, this limits the applicability of the IKC in specialized scenarios.

In summary, the IKC provides a promising foundation to model APT campaigns and attack steps based on its notion of chained stages. However, due to the weaknesses describes above, the original variant is not commonly used in today's analyses and instead is superseded by variations that we describe in the next section.

2.2.2 Kill Chain Variations

In recent years, several variations of the IKC have been proposed to address some of the weaknesses outlined above. These variations either (*i*) address specific challenges for APT detection in selected domains, (*ii*) aim to improve on the overall model by

suggesting additional stages that are generally applicable either from academia or based on analysis of real-world case studies.

Domain-specific Variations

Hahn et al. [Hah+15] propose a security analysis framework for attacks on cyber physical systems (CPSs) based on the *cyber-physical system kill-chain* that (*i*) adds additional stages relevant for CPS and (*ii*) distributes the stages across three layers, namely *cyber*, *control*, and *physical*. The cyber layer focusses on the aspects of regular IT operations and thus is the layer where the complete IKC [HCA11] is located. The control layer describes the industrial control components that are in place throughout the target system including the sensors, actuators, and control algorithms that govern it. The physical layer models the physical process that the target system control including physical properties of the machines and materials used as well as environmental properties such as safety and stability or resource constraints. The first three stages from the IKC (reconnaissance, weaponization, delivery) are modeled across the layers followed by a new phase titled *cyber execution* that encompasses the IKC stages exploitation, C2/C&C and action on objectives. To model the impact the attacker aim to generate on the process, a separate stage the expresses this intent is introduced on both non-cyber layers: *perturb control* on the control layer and *physical objective* on the physical layer. The combination of these three new layers more accurately describes how the attack transitions from cyber across control to the physical layer to achieve the campaign objectives.

The ICS Cyber Kill Chain [AL15] proposed by Assante and Lee in a publication for the SANS institute adapts the IKC [HCA11] to model cyber attacks on industrial control systems (ICSs). The model consists of two stages that are both divided into multiple (sub-)phases that partially map to IKC stages. The first stage is called *Cyber Intrusion Preparation and Execution* and focuses on the attack of the IT-systems of the target network. It encompasses the following sub-phases: planning, preparation, cyber intrusion, management & enablement, and sustainment, entrenchment, development & execution. While the focus of this stage is on the intrusion to the IT-systems, the adversary may also collect information about the ICSs in the network. At the end of this stage, the APT has persisted in the network and starts to access ICSs. The second stage, *ICS Attack Development and Execution*, models the adversary's intentional attacks on the ICSs in the network. This distinction is important as the attacker may have interacted with the ICSs for intelligence gathering purposes resulting in "unintentional attacks". This is due to the fact that ICSs generally are more fragile when faced with unexpected interactions and thus fail easier than IT-systems. The second stage consists of the following sub-phases: attack development & tuning, validation, and ICS attack. The authors highlight that large parts of this stage, specifically attack development and validation, are usually performed offline on dedicated hardware matching the target specifications to avoid detection. Additionally, the difficulty of attacking an ICS is determined based on the type of intended impact with *loss* as the easiest type of impact to achieve, *denial* with moderate difficulty, and *manipulation* as the most difficult type of impact. The publication also includes two case studies that match the proposed two-stage model to prominent APT campaigns that involved ICSs: Havex and Stuxnet. Reports for Havex show a successful stage 1 attack according to the ICS Cyber Kill Chain. The adversaries performed three attacks using spearphishing and the watering hole technique and obtained access beyond the target organization's demilitarized zone (DMZ) to the internal ICS networks. However, as no malware was

delivered to an ICS and thus no ICS attack took place, stage 2 has to be considered failed or unsuccessful. Stuxnet can be characterized as a successful stage 2 attack according to the ICS Cyber Kill Chain. The adversaries gained intelligence on the target networks and ICSs over multiple years. The malware was then first delivered to and distributed in the target networks (including the air-gapped parts) without any ICS payload. The reconnaissance, planning, and delivery sub-phases map to stage 1 of of the model and were successfully executed. Next, the ICS malware was developed, tested offline, and configured to activate once the correct centrifuges were infected. Finally, this payload was delivered using the previously established malware instances. The successful activation of the payloads triggering the physical destruction of the centrifuges marked the end of the campaign and stage 2 of the ICS Cyber Kill Chain.

Kim et al. [KKK19] propose a revised kill chain model for multimedia service environments that focuses internal threats. This new model reflects this by consisting of two levels representing internal and external threats respectively and providing backlinks between certain key stages. This helps to represent the persistent nature of APTs more accurately. Both levels share five stages from the IKC [HCA11], namely reconnaissance, weaponization, delivery, exploitation, and installation. However, the model emphasizes differences in methodology between the internal and external versions of each stage, e. g., external delivery uses sociotechnical techniques while internal delivery is usually implemented via technical means. The authors also highlight how their revised model can help to take countermeasures against internal threats. As these threats can be modeled explicitly instead of being included in the C2/C&C stage of the IKC, defenders can deploy measures beyond simple IP/domain blocking, e. g., endpoint protection/host intrusion detection systems (HIDSs).

Haseeb et al. [HMW20] propose the Internet of Things Kill Chain (IoTKC) model that is derived from the IKC [HCA11], but focusses on details about attack characteristics and attacker behavior specific to IoT devices. The model is based on an empirical study consisting of 300 335 attacks on a medium-interaction honeypot simulating a vulnerable IoT device. The authors first derived five core research questions concerning target identification, initial compromise, command distribution, and goals of attacks on IoT devices with the goal of exploring differences between IoT-based attacks and attacks following the IKC. A dataset consisting of 300 335 attacks from 29 046 distinct IP addresses was then captured over a four-month period via a modified version of the Cowrie honeypot [OT22] to empirically evaluate these questions. The honeypot was configured to listen on 14 ports commonly associated with IoT devices such as remote administration protocols (SSH and telnet), web protocols (DNS, HTTP, and HTTPS) and vendor-specific protocols, e. g., from Amazon Echo or Philips Hue smart bulbs. In their analysis the authors identified nine stages that characterize attacks on IoT devices and form the IoTKC: discovery of devices, entering the devices, getting device information, preparing the device, downloading the package, preparing the package, installing the package, removing traces, and performing actions. While some of the stages are generic to attacks on any networked device, some stages describe key steps specific to IoT intrusions such as the removal of previous malware in the device preparation stage or the establishment of SSH tunnels to use the device for further actions to external targets in the last stage. Overall, the IoTKC represents a specialized kill chain tailored to a single use-case. The paper confirms this by comparing the IoTKC with the IKC and highlighting the core differences. While the IKC helps to generalize a large class of complex attacks, namely advanced persistent threats (APTs), the IoTKC provides a linear model for a smaller subset of attacks on IoT devices.

Generic Model Improvements

Malone [Mal16] proposes an *expanded kill chain* model consisting of three sub-models, namely original IKC [HCA11], an internal kill chain and a target manipulation kill chain. The extended model offers additional depth for modeling the actions on the objectives and host that an adversary performs after the network is initially breached. Overall the model consists of seventeen sub-stages starting from the beginning of the IKC (external reconnaissance) to execution as the last action the adversary performs to achieve their overall goal.

The Bryant Kill-Chain Model [BS17] is a framework to model APT activity for the forensic context that extends existing kill chain models with a relational database to enable data aggregation and improve analysis. The model is divided into four key phases (network, endpoint, domain, and egress) that contain seven stages in total. The authors apply the framework in forensic postmortem analyses to evaluate existing detection and correlation approaches regarding missed stages. Additionally, a proprietary security information and event management (SIEM) system was used to highlight how existing deployments can benefit from including the model in their detection rules.

Other kill chain-based model are based on the analysis of disclosed real-world APT attack reports. Li et al. [Li+16] propose an APT lifecycle model consisting of four high-level stages and eleven sub-stages based on 89 reports obtained from the *aptnotes* repository [BC22]. While the authors argue that their proposed model captures insights from the public APT reports, the concrete choice of stages is not clearly justified based on the surveyed reports. Furthermore, the sub-stages offer little benefit over the existing IKC [HCA11] and simplify the process by only including four high-level stages that are also strictly sequential. Ussath et al. [Uss+16] analyze twenty-two public APT reports and group the malicious activity into a model consisting of three high-level stages, namely *initial compromise*, *lateral movement*, and *C2/C&C*, with ten highly-specific sub-stages. While the resulting model is well suited to the surveyed APT campaigns, it is questionable if it is applicable to other campaigns that might deviate significantly.

2.2.3 MITRE ATT&CK®

The MITRE ATT&CK® framework [The22b] is a knowledge base derived from past real-world cyber attacks that can be used to model potential and specific attack scenarios or threat actors. While it is not specifically tailored or limited to APTs, it is often considered an alternative model to the IKC [HCA11] for modeling APT scenarios. ATT&CK it freely accessible and continuously updated by both MITRE and a surrounding community with bi-annular major releases via GitHub[4]. The latest available release at the time of writing is ATT&CK (for enterprise) v11.3.

The framework is structured into *tactics*, *techniques*, *sub-techniques*, *data sources*, *mitigations*, *groups*, and *software*. This facilitates accurate descriptions of adversaries and their behavior on all required levels of abstraction from high-level goals down to concrete low-level/operational attacks. The following briefly describes each category and provides an example:

Tactics describe the high-level goals attackers aim to achieve such as "Lateral Movement". These abstract goals are often compared to attack stages from kill chain

[4] see: https://github.com/mitre/cti/releases/

models such as the IKC [HCA11]. This similarity is also reflected in the number of tactics present. ATT&CK v11.3 contains fourteen tactics compared to the seven stages of the IKC.

Techniques describe processes how attackers achieve a tactic without specifying concrete tools or exploits. This intermediate representation already includes detailed information such as platform and OSs affected but usually does not reference concrete tools. However, techniques already reference potential mitigation approaches and data sources that can be leveraged for detection. An example technique to achieve the aforementioned "Lateral Movement" tactic is be "Remote Service Session Hijacking". ATT&CK v11.3 contains 191 techniques. Compared to the fourteen tactics, this number already demonstrates the lower level of abstraction.

Sub-techniques describe concrete and detailed implementations of techniques that have been encountered in-the-wild. They can include exploits, known malware strains or information about threat actors that are known to have used the technique in question. Mitigation approaches and detection data sources are also referenced and can be identical to their parent items or more specific to the sub-technique. An example sub-technique for the "Remote Service Session Hijacking" technique is "SSH Hijacking". ATT&CK v11.3 contains 386 sub-techniques.

Data Sources describe elements or components in the network that can provide data for detection with a common example being "Network Traffic". The category is further divided into *Data Components* that reference a domain such as enterprise or ICS and contain backreferences to techniques and sub-techniques. ATT&CK v11.3 contains thirty-nine data sources.

Mitigations describes measures to lower the impact of specific (sub-)techniques. They are usually on a medium abstraction level and do not mention concrete tools similar to techniques. However, they provide more specific guidance linked to the referenced (sub-)techniques. As a consequence the absolute number of mitigations is rather low (43 for ATT&CK v11.3). An examples mitigation measure for the "SSH Hijacking" sub-technique is "Password Policies".

Groups describe clusters of threat activities that are attributed to a common adversary or group of adversaries. They link both (sub-)techniques and software used by the respective group and provide publicly available references for the claims. As the naming scheme between different security vendors is highly diverse, other names for the same group are also mentioned. Examples are the APT groups mentioned in Section 2.1 (APT1, APT28, and APT41) or the "Carbanak" group. ATT&CK v11.3 contains 134 groups.

Software describes concrete applications and tools used by adversaries such as malware and other tools that are either commercial, open-source, built-in, or publicly available. Examples include `PsExec`, `mimikatz`, or the infamous `Stuxnet` malware. ATT&CK v11.3 contains 680 pieces of software.

With its various categories ATT&CK acts as a knowledge base to model adversaries, attack campaigns, and defense measures. However, the framework does not enforce or promote any higher-order modeling strategy. Combined with the fact, that all entities can be accessed by unique identifiers makes the framework a perfect reference that can be embedded into other tool or approaches across the whole defense stack. Other

The Unified Kill Chain		
1	**Reconnaissance**	*Researching, identifying and selecting targets using active or passive reconnaissance.*
2	**Weaponization**	*Preparatory activities aimed at setting up the infrastructure required for the attack.*
3	**Delivery**	*Techniques resulting in the transmission of a weaponized object to the targeted environment.*
4	**Social Engineering**	*Techniques aimed at the manipulation of people to perform unsafe actions.*
5	**Exploitation**	*Techniques to exploit vulnerabilities in systems that may, amongst others, result in code execution.*
6	**Persistence**	*Any access, action or change to a system that gives an attacker persistent presence on the system.*
7	**Defense Evasion**	*Techniques an attacker may specifically use for evading detection or avoiding other defenses.*
8	**Command & Control**	*Techniques that allow attackers to communicate with controlled systems within a target network.*
9	**Pivoting**	*Tunneling traffic through a controlled system to other systems that are not directly accessible.*
10	**Discovery**	*Techniques that allow an attacker to gain knowledge about a system and its network environment.*
11	**Privilege Escalation**	*The result of techniques that provide an attacker with higher permissions on a system or network.*
12	**Execution**	*Techniques that result in execution of attacker-controlled code on a local or remote system.*
13	**Credential Access**	*Techniques resulting in the access of, or control over, system, service or domain credentials.*
14	**Lateral Movement**	*Techniques that enable an adversary to horizontally access and control other remote systems.*
15	**Collection**	*Techniques used to identify and gather data from a target network prior to exfiltration.*
16	**Exfiltration**	*Techniques that result or aid in an attacker removing data from a target network.*
17	**Impact**	*Techniques aimed at manipulating, interrupting or destroying the target system or data.*
18	**Objectives**	*Socio-technical objectives of an attack that are intended to achieve a strategic goal.*

FIGURE 2.1: APT stages according to UKC [Pol17; Pol21]

frameworks, standards, and publications from industry or academia frequently link to certain ATT&CK entities or the framework as a whole. Additionally, MITRE offers integrations with several programming languages or tools such as Bro/Zeek ATT&CK-based Analytics and Reporting (BZAR) [The22a] which provides Zeek [Zee22] scripts to detect certain (sub-)techniques from ATT&CK in network traffic.

2.2.4 Unified Kill Chain (UKC)

The most comprehensive general model to-date is the unified kill chain (UKC) proposed by Pols in 2017 [Pol17] as part of a thesis and subsequently released as a whitepaper in 2021 [Pol21]. The model is based on both a literature review of existing academic models as well as analysis of real-world incident reports of exemplary APT campaigns and thus *unifies* both academic and industry knowledge. This is also reflected in its usage across both industry [Sea21; Sen22] and academia [NDD21; MC21]. *Note*: While the publication uses the terms tactic (as in tactics, techniques, and proceduress (TTPs)) and phase nearly interchangeably to describe a step in the overall kill chain. To align this section with the rest of the thesis, we will use the term (attack) stage.

The UKC consists of three phases encompassing a total of eighteen attack stages as shown in Figure 2.1. While the UKC suggests a rough ordering of stages, it explicitly highlights the diversity in APT campaigns resulting in different orders for each attack. The three stages represent "milestones" or intermediate goals the attacker aims to achieve in the overall campaign. Interestingly, the UKC envisions all three phases as loops that are executed until the attacker achieves the respective intermediate goal with variations and adaptions in both the attack stages that are executed as well as the underlying operational tactics that are used to implement the stage. (*i*) stages 1–9 form the first phase titled *Initial Foothold*. Here the attacker aims to breach the network and gain access to systems and/or data that are part of the trusted environment. This encompasses stages like (initial) reconnaissance, (malware) delivery, social

engineering, and command & control. Loops in this phase represent failed attempts and the following adaptions the attacker performs until the network is breached (or the campaign is aborted). (*ii*) stages 9–14 belong to the second phase titled *Network Propagation*. This phase is only executed if the initially compromised systems do not offer the required data or access the attacker requires to execute their target actions. Common examples are lateral movement (e. g., to a central database server) or credential access and subsequent privilege escalation to increase the attacker's reach into the network. Loops in this phase are executed until the attacker reaches the systems required for the following target actions. (*iii*) stages 15–18 are part of the third phase titled *Action on Objectives* that contains all attack stages that the attacker executes to achieve their overall campaign goal. This includes collection and exfiltration of sensitive data or impact on systems or processes that impede the target organization. Loops in this phase can be expected until the campaign is either discovered or the attacker fulfills an overall goal that does not warrant extended stay in the target network, e. g., destruction of OT components in the case of Stuxnet [Zet14].

The publication further describes how the UKC can be leveraged to describe specific adversaries or attack campaigns by instantiating the UKC as *attacker specific kill chains* or *actor specific kill chains* as also possible with the original IKC [HCA11]. Additionally, the common assumption from the IKC, that all stages must be completed by the attacker to achieve their objective and thus mitigating a single stage is a feasible defense strategy, is challenged by the fact that single stages may be bypassed or skipped (as revealed by the surveyed literature). The authors instead suggests a more in-depth defense strategy that focuses on frequently observed stages and hinders the attacker further by employing network segmentation and zoning. Furthermore, the UKC is compared with the IKC [HCA11] and MITRE's ATT&CK by comparing the stages present in the respective approaches. The comparison emphasizes that the UKC offers the most comprehensive model that is able to model and represent the various aspects of APTs and other high-profile cyber threats.

2.3 TLS Fundamentals

This section gives a brief overview about required background on handshakes and key derivation in the TLS protocol. This is auxiliary information refer to in our approach for passive TLS decryption on network monitoring systems (NMSs) that we present in Section 4.1. *Note*: As of 2022, all version below TLS 1.2 have been deprecated [MF21], as such we will only cover TLS 1.2 [RD08] and 1.3 [Res18].

2.3.1 TLS Handshakes

TLS was designed to protect confidentiality and data integrity between two communication partners. All TLS connections start with a handshake between the two endpoints. In this handshake both endpoints agree on a cipher suite to protect application data, and establish the necessary cryptographic material. To authenticate, the endpoints can also verify each other's identity via X.509 certificates. During a standard TLS handshake only the server authenticates towards the client, while the client authenticity is usually checked on the application-level by other means, e. g., by entering user credential on the website retrieved.

For new connections without any preceding communication a full TLS handshake is performed (as shown in Figure 2.2. Since no prior information is available, both

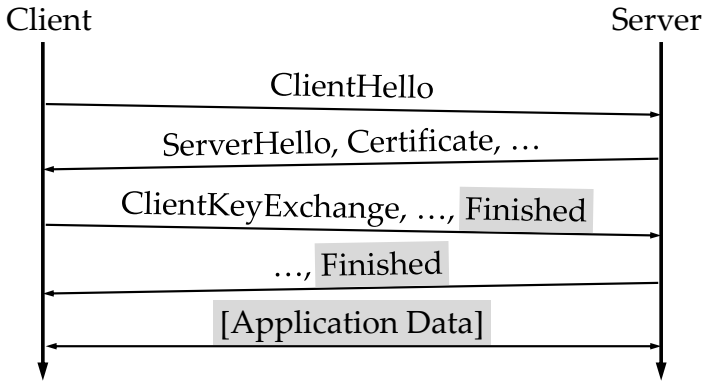


FIGURE 2.2: TLS 1.2: Full handshake (gray indicates encrypted data)

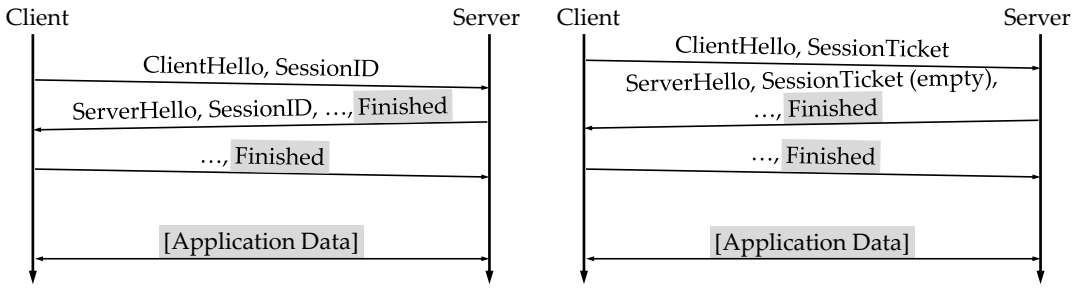
endpoints negotiate a cipher suite and perform a key exchange to establish shared cryptographic material. The exact type of key material depends on the TLS version with TLS 1.2 using (pre-)master secret and TLS 1.3 differentiating between seven different types of secrets including early traffic secrets, handshake secrets and traffic secrets.

Additionally, both TLS versions supports session resumption mechanisms that allow the endpoints to retain some cryptographic material to reduce the operations required to establish subsequent connections. TLS 1.2 offers *session IDs* as well as *session tickets* with the latter containing all cryptographic state to relieve the server from storing this material. The adapted handshakes are shown in Figure 2.3a and Figure 2.3b.

TLS 1.3 supports resumption through a modified session ticket approach based on PSK. The overall concept remains unchanged: (*i*) in full handshakes a key exchange scheme is used to establish a shared secret. (*ii*) resumed handshakes skip this and used established key material in form of a PSK. However, the PSK is altered for every new connection and sent encrypted compared to session IDs and tickets that are both sent in the clear.

2.3.2 TLS Key Derivation

After a shared secret has been established, integrity and confidentiality keys are derived by both endpoints. In the following we describe the process for TLS 1.2 (as shown in Listing 2.1). First, a *master secret* is derived by using the TLS pseudo-random function (PRF) as defined in RFC 5246 [RD08]. This function uses a cryptographic



(A) Via SessionID (B) Via Session Ticket

FIGURE 2.3: TLS 1.2: Resumed handshakes (gray indicates encrypted data)

hash function depending on the selected cipher suite. Next, the PRF is used again to derive up to three sets of secrets containing one secret for server and client, respectively. The encryption keys (`client_key`, `server_key`) are always used. The integrity of the encrypted data is either ensured via a separate HMAC or by using an AEAD cipher that integrates data integrity checks into the decryption process. TLS 1.3 only supports AEAD ciphers and derives keys via HKDF [KE10] instead of TLS PRF but ultimately obtains the same set of keys. Once keys for encryption and integrity protection are available, all following application data in the TLS connection is both integrity and confidentiality protected using the negotiated cipher suite.

```
1  pre_master_secret = <obtained from key exchange>
2  master_secret = TLS_PRF(
3      pre_master_secret,
4      "master secret",
5      client_random + server_random)
6  key_buffer = TLS_PRF(
7      master_secret,
8      "key expansion",
9      server_random + client_random);
10
11 client_MAC = key_buffer[0..31]
12 server_MAC = key_buffer[32..63]
13 client_key = key_buffer[64..79]
14 server_key = key_buffer[80..95]
15 client_IV = key_buffer[96..99]
16 server_IV = key_buffer[100..103]
```

Listing 2.1: TLS 1.2: Key derivation from pre-master secret

2.4 Chapter Summary

This chapter provided background information required for this thesis. We first defined the term advanced persistent threat (APT) based on existing definitions from literature. This definition highlights the problems and challenges that APT detection approaches face when compared with traditional or legacy intrusion detection approaches such as their stealthiness and tendency to remain undetected for large timespans. The next section gave an overview about approaches and frameworks that have been proposed to model APT campaigns and other related activity. Kill chain-based models highlight the interdependencies between different steps such complex attacks with the UKC [Pol21] emerging as the most comprehensive model to date. In the last section, we briefly summarizes information about handshakes and key derivation in (modern) TLS as auxiliary information for our approach to passively decrypt TLS payloads via cooperative endhosts that we present in Section 4.1.

The following chapter establishes key requirements for APT detection in large-scale networks and presents a taxonomy that classifies relevant literature from both industry and academia. Next, all publications are discussed in detail and compared with the established requirements where applicable to identify gaps and motivate our following contributions.

3 Requirements and State of the Art

This chapter defines requirements for effective detection of advanced persistent threats (APTs) and summarizes the state of the art approaches in this area. After the requirements are introduced in Section 3.1, a taxonomy for all relevant areas of the detection process is presented in Section 3.2. The remaining sections discuss and analyze relevant literature in the five main areas covered by the taxonomy: (*i*) event-level monitoring in Section 3.3, (*ii*) alert-level detection in Section 3.4, (*iii*) alert correlation in Section 3.5, (*iv*) APT stage detection in Section 3.6, and (*v*) APT campaign detection and reconstruction in Section 3.7. Lastly, Section 3.8 summarizes the chapter and briefly introduces the following two chapters with the main contributions of this thesis.

3.1 Requirements for APT Detection

APT detection can be seen as a specialized form of intrusion detection and thus shares many of the same requirements for a potential ideal solution. This section describes seven essential requirements for an intrusion detection system (IDS) aimed at APT detection in enterprise networks.

R1: Accuracy The key requirement for any intrusion detection approach is how *accurate* attacks and attack campaigns are detected. This usually entails a high *true positive rate* (*TPR*), i. e., the percentage of attacks that are correctly detected, as well as a low *false positive rate* (*FPR*), i. e., the percentage of benign activity that is wrongly classified as an attack.

R2: Explainability After a security incident has occurred, security operations center (SOC) analysts need to investigate the impact and mitigate potential damages caused by the attackers. This process is massively simplified if the generated alerts are *explainable*, i. e., describe the incident precisely and offer additional context that is required to assess the attacker's actions.

R3: Low overhead To minimize the damage caused by attacks—be it traditional attacks or complex APTs—detection should operate in real-time to detect intrusions as they happen. *Low overhead*, especially in terms of computation, is essential to achieve this, as vast amounts of security-relevant data need to be consumed and timely processed.

R4: Scalability As enterprise networks grow larger, the detection approach needs to be able to sufficiently process the increasing amounts of data from both network traffic and host activity. To achieve this, the approach has to *scale* linearly with its available resources [Hil90].

R5: Security IDSs are often targeted by sophisticated attackers, either by directly compromising the machine or by circumvention via other means. Therefore, an IDS should be *secure* and (*i*) exhibit a minimal threat surface, i. e., make it hard to be compromised by an emergent attacker and (*ii*) strive to give the attacker as little impact as possible if the underlying machine is compromised.

R6: Privacy The data that is used for APT detection is usually extremely sensitive as it contains host and network activity of employees and other users in the organization. Ideally, an IDS should balance the needs of SOC analysts and *privacy* of users by only collecting and processing as much data as required.

R7: Deploy- & Maintainability APTs target large organizations and enterprise networks with large amounts of hosts. At this scale, maintenance of security infrastructure becomes a challenge of its own. Thus, detection approaches should be (comparatively) easy to deploy and maintain in enterprise contexts with minimal impact to overall network operations.

3.2 Classification of State of the Art

Comprehensive APT detection is a complex process across multiple layers of monitoring and detection: (*i*) on the lowest level, *event-level monitoring* provides visibility into host information and network traffic. (*ii*) Above that *alert-level detection* describes processes that strive to generate low-level/high-volume indicators of malicious activity from the underlying events. (*iii*) *Alert correlation* approaches connect and match alerts according to pre-defined rules and/or shared attack characteristics. (*iv*) The area of *APT stage detection* contains detection algorithms specifically tailored to key stages from kill chain-based APT models (as introduced in Section 2.2). (*v*) Approaches for *APT campaign detection and reconstruction* combine information obtained on the lower layers to reveal complete APT campaigns. Figure 3.1 shows a taxonomy of these different research areas with more fine-grained subclasses for the first four areas. The remaining sections of this chapter discuss relevant literature along this taxonomy and compare it with the requirements established in the previous section.

Event-Level Monitoring

The area of event-level monitoring encompasses approaches which aim to obtain visibility into low-level host activity and network traffic & metadata. This layer represents the foundation for intrusion detection in general and APT detection specifically, as data has to be collected and processed to be available for further higher-level analysis and correlation. Most of event-level monitoring is conceptually simple but poses diverse technical challenges. As a consequence, academic research in this area is limited and usually focused on specific problem domains and questions. The two subclasses of literature, that are discussed in Section 3.3, are thus chosen according to the contributions in this thesis, namely:

- Visibility into encrypted network traffic [CHK21; Nay+15; Nay+17; Bie+17; Lee+19]
- Data provenance generation [KC03; KC05; Poh+12; BBM15; Pas+17].

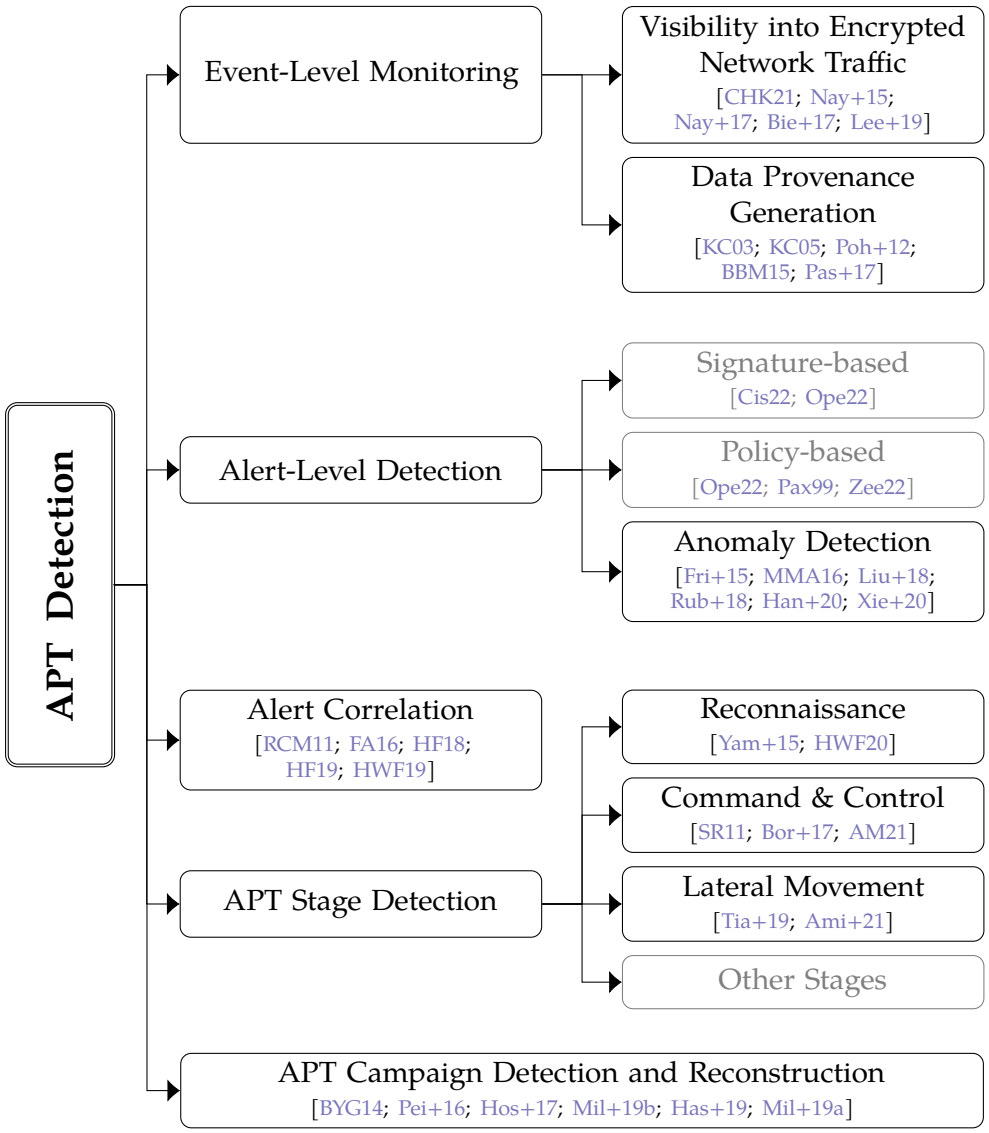


FIGURE 3.1: Taxonomy of APT detection. Gray elements are out of scope for this thesis.

Alert-Level Detection

The area of alert-level detection contains approaches which strive to identify malicious activity present in the generated events. The exact methods vary from approach to approach but usually produce an alert that has a one-to-one mapping to an event, e. g., a single connection or even packet in the network traffic or execution of a binary in the host context. While alerts obtained this way are valuable, they usually cannot detect large and complex attacks or fail to provide sufficient context for proper incident response and mitigation in APT scenarios. Relevant literature is discussed in Section 3.4 and is further divided into the following three subclasses:

- Signature-based detection [Cis22; Ope22] (out of scope for this thesis)
- Policy-based detection [Ope22; Pax99; Zee22] (out of scope for this thesis)
- Anomaly detection [Fri+15; MMA16; Liu+18; Rub+18; Han+20; Xie+20]

Alert Correlation

The area of alert correlation encompasses approaches that link, combine, and otherwise correlate multiple alerts according to common properties. Typical examples are clustering of related alerts that are believed to belong to the same attack or merging of alerts with common metadata such as IP addresses. This step aims to reduce the analysis burden on security experts as clusters can be filtered or triaged more efficiently instead of single alerts. In APT contexts, classic alert correlation can still be used to achieve this goal, but usually falls short to correctly cluster stealthy attack steps employed by the attacker as part of an overarching APT campaign. As this research area is rather large, exemplary relevant works are discussed in Section 3.5.

APT Stage Detection

The area of APT stage detection contains approaches that aim to detect a single stage from a kill chain-based APT reference model such as the intrusion kill chain (IKC) [HCA11] or the unified kill chain (UKC) [Pol17; Pol21] (see: Section 2.2). While this is usually not sufficient to completely reveal or prevent APT campaigns, detection of a single stage can aid in reconstruction efforts (as presented in the next taxonomy class). This is especially true when multiple approaches are used in conjunction. Relevant literature is discussed in Section 3.6 and covers three essential APT stages (as identified by Ussath et al. [Uss+16]):

- Reconnaissance [Yam+15; HWF20]
- Command & control [SR11; Bor+17; AM21]
- Lateral movement [Tia+19; Ami+21]

APT Campaign Detection and Reconstruction

The area of APT campaign detection and reconstruction encompasses approaches that strive to achieve the same overall goals of this thesis of comprehensive detection of entire APT campaigns. As a consequence, the requirements presented earlier in this chapter, are highly applicable to these approaches. They usually leverage an underlying kill chain-based APT model (see: Section 2.2) to reconstruct indicators of single stages or other malicious activity towards a full campaign view that is aimed at supporting human SOC analysts or threat hunters.

3.3 Event-Level Monitoring

This section discusses relevant approaches for event-level monitoring which provide the foundation for all higher-level intrusion detection. The goal of these approaches are to generate visibility in both network and host contexts that can then be used for further analysis. As this area is quite applied and not generally well represented in academia, the two relevant subclasses are directly relevant for contributions made in this thesis. Approaches in this taxonomy class are further structured into two subclasses (see: Section 3.2): (*i*) visibility into encrypted network traffic and (*ii*) data provenance generation. It is also important to note that both *R1: Accuracy* and *R2: Explainability* are not directly applicable to approaches in this section and are thus skipped in the discussions.

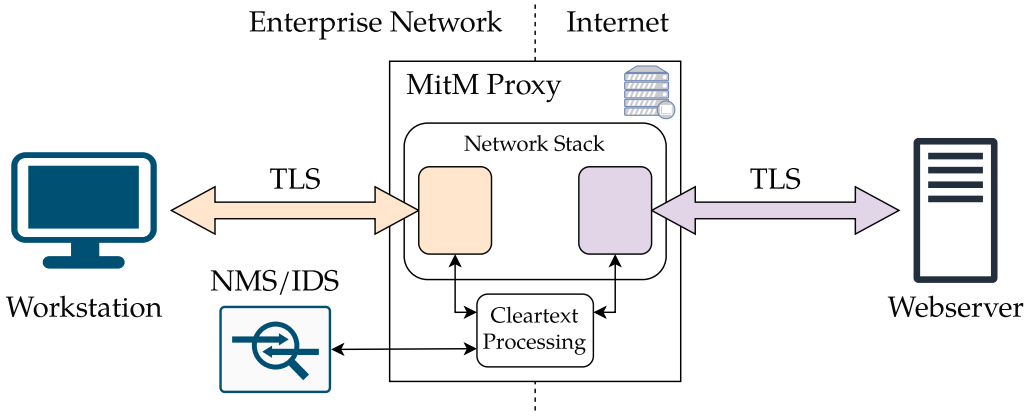


FIGURE 3.2: Overview: TLS interception via MitM proxy servers

3.3.1 Visibility into Encrypted Network Traffic

Encrypted network protocols are increasingly popular and beneficial to overall security in both the Internet and private networks. TLS is especially prevalent even in purely network-internal communication to protect against potential insider attackers with access to the network. This poses problems to monitoring systems as they can no longer directly access the cleartext of connections and need to fall back to metadata-only analysis. This section presents approaches that aim to restore this capability to network monitoring systems (NMSs) and network-based IDSs.

TLS interception via Man-in-the-Middle (MitM) proxy servers such as `mitmproxy` [CHK21] (sometimes also referred to as "split" TLS) is the predominant approach to obtain access to cleartext traffic for security monitoring in enterprise networks. Figure 3.2 shows a typical deployment where the proxy server is deployed at the network edge to act as the central gateway to Internet servers. Client hosts inside the enterprise network (e. g., workstations) are mandated via policy and network configuration to only access the Internet via this gateway such that all cleartext traffic can be passed to an NMS/IDS adjacent to the proxy. Additionally, the proxy can perform other kinds of analysis on the cleartext traffic, e. g., filtering, or compliance checks.

This approach works decently in enterprise environments where network policy can enforce proxy usage across clients and security concerns take precedence over privacy concerns. However, it has several problems as outlined in the literature [Dur+17]: (*i*) The certificate authority, the proxy uses to generate certificates for the visited domains, needs to be fully trusted by all endpoints in the network. As the proxy essentially masquerades as the target server, it needs to generate certificates that the endpoint accepts as valid. This is a potential threat vector as certificate authority would enable an attacker to arbitrarily MitM any connection if they obtain the associated private key. (*ii*) TLS interception breaks mutual authentication as the client endpoint does not see the actual server endpoint and also cannot use its own certificate towards the server. Additionally, practices like certificate pinning break as the proxy cannot offer the pinned certificate. (*iii*) Performance is suboptimal as the proxy needs to establish and maintain two separate TLS connections per original connection. This overhead is even worse for typical web scenarios where short-lived connections are used to retrieve assets of a web page as the initial setup burden imposed by the

handshakes accumulates on the proxy.[1] (*iv*) The security infrastructure like NMS/IDS conceptually obtain access to all payloads even for special cases where this would be unwanted or even not permitted such as medical contexts. While many enterprise settings may favor absolute visibility over user privacy, this is certainly not true for all contexts. However, as the proxy intercepts all TLS connections, users have to trust the network operators that sensitive data (like online banking) is not forwarded.

Based on the two TLS connections that need to be established for each middlebox/proxy, *R3: Low overhead* is not fulfilled. Similarly, *R4: Scalability* is also not met as the centralized aspect of TLS interception hinders horizontal scaling. *R5: Security* is not checked as each middlebox/proxy gains full read/write access to all cleartext payloads thus massively impacting the overall security of the approach. Furthermore, clients have to place absolute trust in the certificate authority, thus making this a very attractive target for attackers. The unlimited access to cleartext payloads also prevents *R6: Privacy* from being checked. Lastly, *R7: Deploy- & Maintainability* is party checked as the configuration of clients is suboptimal but manageable. Overall, TLS interception is not a good solution to our original goal of obtaining visibility (i. e., read-only access) into encrypted communication.

Multi-Context TLS (mcTLS) [Nay+15] aims to support in-network functionality like performance monitoring or intrusion detection in TLS as an extension to the standard that attaches so-called contexts to the protocol's header. These contexts allow the endpoints to explicitly grant fine-grained access to different parts of the transmitted payloads (e. g., HTTP header or body) to middleboxes.

The design of mcTLS was guided by five key requirements: (*i*) *entity authentication*, i. e., authentication of all communication partners including middleboxes, (*ii*) *data secrecy* as targeted by standard TLS, (*iii*) *data integrity & authentication* i. e., detection of modifications including the distinction between communication endpoint and trusted middlebox, (*iv*) *explicit control & visibility*, i. e., both endhosts must consent to allow a new middlebox to the connection, and (*v*) principle of *least privilege* by only granting middleboxes the minimum level of access to the payloads required to perform their task. Additionally, the protocol should not impose significant overhead to either performance or maintenance.

Based on these requirements, mcTLS extends TLS by adding a context byte to the record format and an extended handshake sequence that ensures that the respective middleboxes obtain two halves of their respective context keys from both endpoints. This results in each entity (endpoints and middleboxes) with a set of contexts and keys that can then be used to encrypt and decrypt the mcTLS records of the respective context. This achieves the requirements as long as no participant shares key material out of band (a scenario that also applies to standard TLS) or multiple middleboxes collude. The context can then be used to model higher-level application behavior either by using contexts for different sections of the payload or by using it as a permission system and systematically placing parts of the payload in the appropriate context. The authors discuss several application scenarios including intrusion detection and HTTP/2 streams.

The authors implement a prototype of mcTLS on-top of TLS 1.2 by modifying the *OpenSSL* library [TYH22] in version v1.0.1j. The prototype is then evaluated in four aspects across one lab environment on a single machine and a wide-area environment

[1] For an expanded description of handshakes in modern TLS refer to Section 2.3.1.

between three Amazon EC2 instances across the globe. The first set of experiments focuses on *time overhead* and separately investigates handshakes, file transfers, and page loads in via HTTPS. Their results indicate that mcTLS does not substantially impact real-world scenarios. The second set of experiments focuses on *data volume overhead* and reveals that mcTLS introduces less than 2% additional overhead compared to standard TLS. The third set of experiments concerned *CPU overhead*. Their results indicate that server performance decreases by 23–25% while middlebox performance increases by 45–75% due to the reduced number of handshakes compared to TLS interception. The last experiment evaluates *deployment overhead* by modifying the Ruby SSL library as well as *OpenSSL* benchmarking facilities to estimate the effort required to port existing applications to mcTLS. Although the selection of applications is limited in some aspects, the results show that only few lines of code are required (10–250 depending on the scenario).

mcTLS does not fulfill *R3: Low overhead* due to the significant latency introduced by the modified handshake. *R4: Scalability* is also not met as the additional computations required by server limit the effectiveness of this approach in enterprise settings. *R5: Security* and *R6: Privacy* are partially met as middleboxes are separately authenticated and are encouraged to use the principle of least privilege. Lastly, *R7: Deploy- & Maintainability* is not met as mcTLS is not incrementally deployable and thus unlikely to ever reach mass adoption.

Middlebox TLS (mbTLS) [Nay+17] it an approach to support trusted middleboxes (running on potentially untrusted hardware) in a TLS connection that is backward compatible, i. e., inter-operates with standard TLS implementations. This is realized by leveraging trusted computation technologies like Intel's Software Guard Extensions (SGX) to ensure that the middlebox runs trusted application code and the key material does not leave the secure enclave.

The protocol is carefully designed based on an analysis of the overall design space of encrypted multi-party communication that the authors discuss first. They conclude that achieving all desired properties is extremely difficult and thus focus on three core requirements: (*i*) *immediate deployability* by backward compatibility with standard TLS, (*ii*) *protection for outsourced middleboxes*, i. e., measure to protect payloads against potentially untrusted middlebox infrastructure, and (*iii*) *middlebox accountability*, i. e., verification of middlebox application code to increase trust for endpoints. As expected, a minimal amount of added overhead is also an implicit goal.

To achieve these requirements, mbTLS adds a `MiddleboxSupport` TLS extension (to indicate support for the protocol by both communication endpoints) and two new TLS record types, namely `MiddleboxAnnouncement` (for optimistic registration of server-side middleboxes) and `Encapsulated` (to wrap TLS records exchanged between one endpoint and one of their middleboxes). The standard TLS handshake is extended in a backward compatible way with optional additional handshakes between endpoints and their respective middleboxes. This ensure that connections consisting of one mbTLS endpoint can still make use of middleboxes even with the other endpoint running standard TLS. Additionally, the handshake is extended with optional remote attestation (to verify middlebox code) and payloads are encrypted with unique per-hop keys to ensure that a network-based attacker cannot learn if a middlebox modified payloads.

The authors evaluate mbTLS in two ways: (*i*) a *security analysis* is performed that qualitatively describes how the protocol fulfills the respective security requirements. (*ii*) a set of real-world experiments demonstrates that it is immediately deployable and evaluates the overheads imposed by the added security benefits. For this, a prototype implementation of mbTLS based on *OpenSSL* [TYH22] in version v1.1.1-dev and the Intel SGX SDK is built as well as an example middlebox function in form of an HTTP proxy performing header insertion. Their interoperability results indicate that mbTLS handshakes are not dropped and their client can connect to the vast majority of Alexa top 500 websites that support HTTPS. For their performance evaluation, the authors compare mbTLS handshakes in various middlebox configurations through a micro-benchmark with both standard TLS as well as "split" TLS (i. e., interception via MitM proxy). Their results indicate that the protocol does increase the computation time with an increasing number of middleboxes (as expected due to additional secondary handshakes) but improves handshake time for the middlebox (as it only needs to do one additional handshake compare to two for "split" TLS). However, real-world experiments show, that overall handshake latency is only increased on average by 0.7%. In the last experiment, the authors evaluate the impact on network performance imposed by using the secure enclave. Their results indicate that processing inside the secure enclave does not have a noticeable impact on overall throughput which is likely the result of overall high interrupt rates in both scenarios (with or without enclave).

mbTLS partially meets *R3: Low overhead* based on the results obtained in the runtime performance evaluation as the performance impact from using the secure enclave is negligible. Similarly, *R4: Scalability* is also partially fulfilled as the approach scaled decently with additional middleboxes added to the connection. *R5: Security* and *R6: Privacy* are also both partially checked as the middleboxes are explicitly added and execution in the secure enclave increases trust in the running code. Lastly, while mbTLS improves upon mcTLS in terms of backward compatibility, the hard requirement of a secure enclave does not fulfill *R7: Deploy- & Maintainability*.

Locally Operated Cooperative Key Sharing (LOCKS) [Bie+17] is an approach to enable trusted middleboxes in an enterprise setting to obtain TLS session keys from endhosts in the network to decrypt and analyze the encrypted payloads. In comparison to MitM proxy servers, this leaves the security guarantees of TLS intact as the connection is not intercepted or otherwise modified. The evaluation in a real-world setting shows promising results for both the added overhead imposed by the decryption and the added latency.

The architecture of LOCKS is composed of (*i*) *client endpoints* inside the enterprise network, (*ii*) *trusted agents* that cache and provide TLS session keys, and (*iii*) *middleboxes* that perform NMS or IDS functionality by passively receiving encrypted traffic and obtaining session keys from the trusted agent and subsequently decrypting and analyzing decrypted payloads. After the TLS handshake is completed by the client, it forwards the newly established session keys to the trusted agent and pauses the connection pauses the connection until keys are stored in the trusted agent. This ensures that keys are received but adds latency to the connection.

The authors implement a prototype of LOCKS consisting of two core components: the client is represented by a modified Firefox browser [Fir22] with a modified NSS library [NSS22] that implements the key sharing and pause logic. The IDS component is implemented in Bro [Pax99; Zee22]. Additionally, a trusted middlebox is implemented but not closely described. Unfortunately, no implementation is publicly

available although all underlying libraries and tools are open-source. The evaluation is divided into three parts: (*i*) the impact on download latency is measured in both local and cloud scenarios and compared with unmodified TLS and a MitM scenario with active TLS interception. The results indicate that the download latency between LOCKS and MitM are insignificant compared to regular variations in network behavior. (*ii*) the authors conducted a user study on an unspecified number of "alpha users" to measure usability. Based on the System Usability Scale (SUS) [Bro96], the results show a normalized usability percentage of 97%. (*iii*) the impact on the IDS induced by adding TLS decryption is measured through packet loss. The results indicate a moderate overhead that can be addresses by increasing the computational resources allocated for the IDS.

LOCKS partially fulfills *R3: Low overhead* as the overhead is limited to client machines needing to forward key material and briefly pausing the connection until receiving a confirmation. Similarly, the approach should scale decently well as the middleboxes can be scaled up horizontally, although this was not explicitly tested. *R4: Scalability* is thus partially checked. Both *R5: Security* and *R6: Privacy* are fulfilled as the endhosts remain in control of the key material and the authors even briefly discuss the possibility of selective decryption. Lastly, *R7: Deploy- & Maintainability* is partially met as the deployment involves updates to endhost software which can be complex in enterprise settings.

Middlebox-aware TLS (maTLS) [Lee+19] is TLS extension aimed at auditing middleboxes by explicitly adding them to the TLS connection via so-called *middlebox certificates* that are additionally logged in middlebox transparency logs. maTLS builds upon these auditable middleboxes and splits the connection from client to server into segments between each endpoint and middlebox respectively. This is effectively a variant of MitM proxies/"split" TLS with additional auditing measures. The authors verify the security goals of maTLS via Tamarin [Mei+13] and also evaluate a prototype implementation for various performance metrics.

maTLS extends the TLS 1.2 handshake by including the `Middlebox_Aware` extension to the `ClientHello` message. This allows each middlebox to insert their own segment keys (on the way to the server) and their own certificates (on the way back to the client) into the respective `*Hello` messages. Additionally, each middlebox establishes an *accountability key* with each of the two connection endpoints that allows them to audit the middleboxes participating in the connection. After the handshake, key material is established for each connection segment in the TLS connection and both endpoints obtained key material and certificates from all middleboxes. Afterwards, data can be transmitted as usual with each middlebox being able to read and write on the payloads (depending on the capabilities indicated in their certificate).

The evaluation consists of two sections and a discussion: (*i*) The security properties of maTLS are evaluated using Tamarin, an automated verification tool built on logic formulae that represents the protocol as a series of states which is a multiset of valid facts. The analysis shows that maTLS satisfies the intended security goals with the Tamarin implementation being openly available. (*ii*) A prototype based on OpenSSL [TYH22] is evaluated for HTTP load time, data transfer time, integrity verification time, and CPU processing time. Their results across three testbeds indicate that maTLS adds about 10–32ms of delay to HTTP loads, that verification takes negligible time, and that both verification time and overall CPU time in the handshake scales linearly with the number of middleboxes as expected. (*iii*) The authors discuss several

Requirement	TLS Interception	mcTLS [Nay+15]	mbTLS [Nay+17]	LOCKS [Bie+17]	maTLS [Lee+19]
R1: Accuracy	— not applicable —				
R2: Explainability	— not applicable —				
R3: Low overhead	✗	✗	○	○	✗
R4: Scalability	✗	✗	○	○	○
R5: Security	✗	○	○	✓	✓
R6: Privacy	✗	○	○	✓	○
R7: Deploy- & Maintainability	○	✗	✗	○	○

TABLE 3.1: Requirement Comparison: Visibility into Encrypted Network Traffic
✓ := fully met ○ := partially met ✗ := not met ◌ := unknown/neutral

aspects of maTLS, namely potential incremental deployment, mutual authentication, abbreviated handshakes, and compatibility with TLS 1.3.

As maTLS effectively builds upon regular TLS interception with explicit middlebox certificates, *R3: Low overhead* is not met. *R4: Scalability* is partially checked due to the option of horizontally scaling middleboxes if required. The addition of explicit middlebox transparency logs and accountability checks *R5: Security*. *R6: Privacy* is improved compared to regular TLS interception due to the addition of capabilities. However, endhosts have no way to conceptually prevent middleboxes from reading selected connections, thus only partially fulfilling the requirement. Lastly, *R7: Deploy- & Maintainability* is also partially met as the deployment is similar to regular TLS interception with some additional efforts for the transparency log.

Summary Table 3.1 gives an overview about the requirements fulfilled by the approaches for visibility into encrypted network traffic discussed in this section. As the table shows there is no approach that fulfills all our requirements although *R1: Accuracy* and *R2: Explainability* are not applicable to this area of research as discussed in the introduction. The results also reveal that approaches that extend TLS in a non-backward compatible way (like mcTLS [Nay+15]) or require specific hardware (like mbTLS [Nay+17]) are highly unlikely to fulfill *R7: Deploy- & Maintainability* or reach adoption. While all approaches improve over the predominant practice of TLS interception, LOCKS [Bie+17] looks most promising when we consider our requirements. While it still has some problems related to the delay induced by pausing the connection until key material is transmitted, its architecture is promising. For this reason, LOCKS served as inspiration when we designed our approach for passive TLS decryption at the NMS that we present in Section 4.1.

3.3.2 Data Provenance Generation

Reliable monitoring of host activity is another important data source for successful APT detection, as large parts of the campaign are local to compromised hosts and do not produce any network traffic (see: Section 2.2.4). Recently, (whole-system) data provenance has been established as state-of-the-art for host activity monitoring due

to its comprehensive tracking of fine-grained actions on kernel-level. This section presents academic approaches that aim to efficiently and reliably capture and process graph-based data provenance.

BackTracker [KC03; KC05] aims to detect an intruder's point of entry into a compromised system by backtracking from the detection point to potential entry points or vulnerabilities. It is one of the first works that analyzes kernel-level objects (e. g., processes, files, sockets…), events (e. g., read, write, connect…), and their interconnections as a graph. While the term *data provenance* is not used, King et al. employ a graph format that is similar to the later established *provenance graphs*.

BackTracker consists of two components: (*i*) an on-line component called EventLogger, that collects kernel-level events on monitored hosts and (*ii*) an off-line component called GraphGen, that builds a dependency graph from the detection point of an observed attack. Both components were implemented as prototypes for Linux. EventLogger leverages a VM monitor to collect event data about the VM guest on the VM host. GraphGen offers five rules to filter and prioritize events to reduce the complexity of the dependency graph, e. g., ignoring certain high-volume folders and files such as `/tmp` or special handling of helper processes that introduce common patterns in the generated graph.

The authors evaluate BackTracker on three real attacks that were captured on a honeypot system running Red Hat Linux 7.0 and one simulated attack. The evaluation covers the *effectiveness* of the proposed filtering rules used by GraphGen as well as the (storage) space and (computation) time *overhead* imposed by collecting kernel-level events via EventLogger. Their results indicate that the filtering rules strongly reduce the size of the dependency graph to about 2.39%–0.28% of the unfiltered graph with absolute sizes of 20–56 objects and 25–81 events. Furthermore, overhead varies greatly between scenarios: space overhead was measured between 0.002 GB/day and 1.2 GB/day while the data collection imposed a computational slowdown between 0% and 9%. Overall, this makes BackTracker feasible for real-world deployments especially on modern hardware.

Hi-Fi [Poh+12] is a kernel-level module that is able to collect high-fidelity whole-system provenance data on Linux hosts. Based on the Linux Security Modules (LSM) framework, Hi-Fi induces only a low overhead between 1% and 6% depending on the workload. The approach is able to capture complete provenance traces from early kernel initialization to system shutdown as the first of its kind.

Hi-Fi consists of three distinct components: (*i*) the *provenance collector* is a LSM module on kernel-level that subscribes to kernel events, (*ii*) the *provenance log* is a buffer at the boundary between kernel- and user-space that receives events from the collector and offers them to user-space as a file, and (*iii*) the *provenance handler* is a user-space program that accesses the file offered by the log and processes or stores the resulting provenance records.

The evaluation focuses on both *correctness* (by executing malicious actions on a monitored host and inspection of the generated provenance trace) and *performance* (by measuring per system-call overhead in various scenarios). Their results indicate that malware behavior is accurately present in the generated provenance traces while incurring an overhead of about 3% in "representative workloads". Overall, Hi-Fi represents a strong baseline for Linux-based whole-system provenance capture as

the authors explicitly did not investigate further application scenarios for provenance data.

Linux Provenance Modules (LPM) [BBM15] is a framework to facilitate trusted and secure (whole-)system provenance capture on Linux. The publication contains three core contributions: (*i*) the authors present Linux Provenance Modules (LPM) and discuss its design and threat model, (*ii*) LPM is leveraged to build a secure provenance-aware deployment by porting Hi-Fi [Poh+12] and supporting SPADE [GT12] (two existing provenance environments), and (*iii*) the authors introduce and implement a scheme for provenance-based data loss prevention to highlight one useful application scenario of trustworthy provenance data.

The underlying threat model of LPM is a provenance-aware host (i. e., a host which has a provenance monitor installed and running) that has been compromised by a remote attacker who might want to add, remove, or otherwise modify the captured provenance data either directly on the host or in transit to long-term storage. LPM now aims to protect a trusted computing base (TCB) consisting of three components: the kernel mechanisms for provenance capture, the provenance recorder (in user-space), and the (potentially remote) storage backend. Additionally, the authors define five system goals for secure provenance monitors based on McDaniel et al. [McD+10] and other past work in the area. They should be: (*i*) *complete* (leave no gaps in the record of system activity), (*ii*) *tamperproof* (TCB cannot be disabled or modified from user-space), (*iii*) *verifiable* (LPM itself and user-attestable kernel), (*iv*) be transmitted over an *authenticated channel*, and (*v*) enable *attested disclosure* for low integrity provenance from user-space applications.

To achieve these goals, LPM is implemented as a parallel framework to LSM to inherit some formal assurances. Similar to LSM hooks, LPM adds *provenance hooks* that enable provenance modules to obtain provenance data from system calls. Additionally, NetFilter hooks are used to implement a cryptographic message commitment protocol based on Digital Signature Algorithm (DSA). The signature is placed in the IP options field and can then be verified on a remote provenance-aware host to ensure continuity of the provenance log. The authors implemented two provenance modules using the provided hooks: *ProvMon* (an extended version of Hi-Fi [Poh+12]) and a modified version of SPADE [GT12]. Next, the publication describes how a system configured with a trusted platform module (TPM) can be configured to establish an attestable chain of trust for a kernel running LPM including support for integrity checking of provenance-aware applications in user space. After the system has booted, the runtime integrity of the TCB can be retained via standard utilities like SELinux.

The evaluation consists of three parts: (*i*) a security analysis confirms a secure deployment as described prior using ProvMon and LPM fulfills the five system goals. (*ii*) a scheme and corresponding implementation of provenance-aware data loss prevention (in form of a file transfer application) demonstrates a novel application scenario for whole-system data provenance. (*iii*) a performance evaluation indicates that LPM offers comparable performance in both capture overhead and space requirements to previous provenance approaches while delivering additional security assurances.

CamFlow [Pas+17] is an approach for whole-system provenance capture that aims for broad adoption especially in PaaS environments. It addresses several shortcomings that prevented previous proposals from gaining significant traction and is evaluated via three application scenarios: compliance, fault/intrusion detection, and data loss

prevention. Its design takes inspiration from PASS [Mun+06], LPM [BBM15], and Hi-Fi [Poh+12] with authors from the first two approaches also appearing on this publication. Based on that, the publication defines four design goals for CamFlow: (*i*) easy maintenance, (*ii*) usage of existing kernel mechanisms, (*iii*) no duplication of existing mechanisms, and (*iv*) potential for integration into the mainline Linux kernel. To achieve this, CamFlow leverages LSM and NetFilter hooks as well as `relayfs` [Zan+03] to efficiently transfer provenance data from kernel to user space.

CamFlow's features set includes several key features from previous approaches. Cross-host provenance is possible by tracking incoming packets through an LSM hook and outgoing packets via a NetFilter hook and including them as entities in the provenance graph. A hardware root of trust can be supported through usage of TPMs and can be verified by remote attestation. System provenance can be augmented by additional application-level provenance. In the presented concept this is realized by ingesting log files (e. g., from webservers or databases) and attaching the application activity to the respective items in the provenance graph. A novel feature introduced by CamFlow is *selective provenance* via a fine-tuned configuration file. While whole-system provenance capture is possible, users can tailor the provenance capture (and thus the volume of generated data) in several ways including defining "object of interest" such that any activity involving them is captured or marking objects as *opaque* such that any activity involving these objects is discarded. Objects can be selected by path name, network address (and port), LSM security context, `cgroups`, or user and group IDs. Alternatively, the configuration can also limit provenance capture to a subset of system calls.

The authors evaluate CamFlow in regard to *maintainability* and *performance* as well as the impact of selective provenance on data volume. Maintainability was mainly accessed through LoC compared to previous two approaches (PASS [Mun+06] and Hi-Fi/LPM [Poh+12; BBM15]) as well as the changes required to port CamFlow between Linux kernel versions. The results indicate that CamFlow is significantly more self-contained and thus interacts with the kernel in clearly defined interfaces. Compared to PASS and LPM which were never ported to newer kernel releases, CamFlow often required no changes at all for new releases (especially between minor kernel versions). The performance evaluation is twofold and contains micro-benchmarks for selected system calls as well as a macro-benchmark simulating realistic system activity. The results indicate that CamFlow imposes overhead in the same order of magnitude that its predecessors with up to 22% overhead in worst case. Furthermore, an exemplary evaluation with selective provenance highlights that tailoring the provenance output can reduce data volume by about 20% compared to whole-system provenance capture. Next to the quantitative evaluation, the authors discuss four application scenarios for data provenance and how CamFlow can support those, namely compliance, intrusion detection, data loss prevention, and retrofitting of existing applications. While CamFlow generally supports all scenarios, data loss prevention especially profits from selective provenance and the taint mechanism to both reduce data volume and perform the actual data loss check in constant time. Overall, CamFlow is a promising option for whole-system and selective provenance that is likely to emerge as one of the default provenance systems.

3.4 Alert-Level Detection

This section discusses relevant approaches for alert-level detection, i. e., approaches that aim to identify malicious activity present in the low-level events. One alerts usually relates to a single event such as a network packet or connection or a single host activity. As no other context across multiple alerts is available, the information added via the alert is usually limited and the volume (and FPR) of these alerts is rather high. Nonetheless, they represent an important part of the overall APT detection process as alerts can be produced for single malicious actions as part of a larger campaign. Approaches in this taxonomy class are further structured into three subclasses (see: Section 3.2): (*i*) signature-based detection, (*ii*) policy-based detection, and (*iii*) anomaly detection. The first two subclasses are usually not capable of reliably detecting traces of APT activity and are thus only covered very briefly in this thesis.

3.4.1 Signature- and Policy-based Detection

Signature- and policy-based detection are conceptually simple approaches to generate alerts from events. Signatures describe patterns or fingerprints of previously detected malicious activity that can be used to efficiently detect the same threats again such as checksums of malicious executables or recurring byte-patterns in botnet traffic. As they are usually static, they offer limited to no protection against new threats (as commonly seen in APT campaigns) and may even change for known threats, e. g., if new compiler version are employed by the attackers that change the checksum of the executable. Policy-based detection aims to identify malicious behavior as deviations from manually established policies such as allowed communication patterns. While this is conceptually similar to anomaly detection, traditional policies are "handwritten" and prone to under- or overspecifying. This section briefly highlights exemplary tools for signature- and policy-based detection in network contexts, namely Snort [Cis22], Suricata [Ope22], and Zeek [Pax99; Zee22].

Snort [Cis22] is an open-source (dual licensed: GPL-2 and non-commercial custom) intrusion prevention system (IPS) mainly developed by Cisco that focuses on rule-based alerting. It has three main operation modes: (*i*) as a packet sniffer that prints information to the terminal (similar to `tcpdump` [Tcp22]), (*ii*) as a packet logger that writes observed packets to storage, or (*iii*) as a rule-based IDS that optionally can stop packets inline (IPS).

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (sid:123; rev:1;
  msg:"HTTP POST request with 'malware' in URI"; fast_pattern;
  flow:established, to_server; http.method; content:"POST";
  http.uri; content:"malware"; classtype:bad-unknown;)
```

Listing 3.1: Snort rule for outbound HTTP POST requests with "malware" in its URI

Snort rules follow a simple format that specifies the addresses, ports, and protocols that a packet has to match as well as certain protocol specific information such as HTTP method or payloads. Listing 3.1 shows an example of such rules that generates an alert for all outbound HTTP POST requests to URIs that contains the keyword "malware". While rules can be written manually, Snort offers two rule sets maintained by Cisco Talos that catch known threats. The *Community Ruleset* is freely available, maintained by the community and receives only QA by Cisco Talos. The *Snort Subscriber Ruleset* is available as a subscription and developed and tested by Cisco Talos.

Suricata [Ope22] is an open-source (GPL-2 license) NMS/IDS/IPS primarily developed by the Open Information Security Foundation (OISF). Its feature set roughly covers three categories:

- Protocol logging
- Rule-based alerting
- Lua-based scripting interface for advanced analysis tasks

Suricata ships with a protocol detection engine similar to Zeek [Zee22] and attaches protocol information to its JSON-based Extensible Event Format (EVE) output format. This is useful for security and non-security use cases alike such as performance monitoring or network-based intrusion detection. Additionally, Suricata is able to optionally extract and store TLS certificates from observed handshakes as well as files that were transmitted through any of its supported protocols.

```
alert http $HOME_NET any -> $EXTERNAL_NET any (sid:123; rev:1;
  msg:"HTTP POST request with 'malware' in URI"; fast_pattern;
  flow:established, to_server; http.method; content:"POST";
  http.uri; content:"malware"; classtype:bad-unknown;)
```

Listing 3.2: Suricata rule for outbound HTTP POST requests with "malware" in its URI

Next to its pure monitoring capabilities, Suricata is well-known for fast and efficient rule-based alert generation. Listing 3.2 shows an example of such rules that generates an alert for all outbound HTTP POST requests to URIs that contains the keyword "malware". Although remarkably similar to the same rule for Snort (Listing 3.1), the Suricata rule is more flexible, as `http` can be directly specified as the protocol while the ports can be set to `any` due to Suricata's protocol detection capabilities. Rules can be loaded from a repository hosted by the OISF via the bundled `suricata-update` utility or written manually and loaded via the configuration file. The repository contains many well-known rulesets like the "Emerging Threats Open Ruleset"[2] or rulesets from abuse.ch[3] or SecureWorks[4].

For use cases beyond rule matching, Suricata offers a Lua scripting interface to perform advanced analysis tasks. A valid script defines two functions (`init` and `match`) and can access various parts of the current packet or connection in so-called "buffers" to perform detection tasks. At the time of this writing, Suricata supports twenty-three buffers including protocol-specific ones like DNS, HTTP, SMTP, SSH, and TLS as well as accessors for raw payloads. Scripts then define which buffers they require in their `init` function and can access them in `match` to perform the analysis task at hand. Finally, the return value of `match` determines if an alert is raised.

Zeek (formerly known as Bro) [Pax99; Zee22] is an open-source (BSD license) passive NMS with a Turing-complete scripting language that allows for flexible deployments. This flexibility makes it appealing to several network-based problem fields such as performance and QoS monitoring, protocol debugging, and naturally, network-based intrusion detection. It produces detailed log files for all supported 50+ supported protocols across both transport and application layer. Listing 3.3 shows the well-known `conn.log` that contains details of all observed TCP and UDP communication.

[2] see: https://rules.emergingthreats.net/
[3] see: https://abuse.ch
[4] see: https://secureworks.com

```
{
  "ts": 1658153792.723982,
  "uid": "CFJ2mL343OmjIYRUnl",
  "id.orig_h": "10.0.1.13",
  "id.orig_p": 40408,
  "id.resp_h": "40.114.177.156",
  "id.resp_p": 443,
  "proto": "tcp",
  "service": "ssl",
  "duration": 0.10236883163452148,
  "orig_bytes": 679,
  "resp_bytes": 254,
  "conn_state": "SF",
  "missed_bytes": 0,
  "history": "ShADFadRfR",
  "orig_pkts": 6,
  "orig_ip_bytes": 975,
  "resp_pkts": 4,
  "resp_ip_bytes": 470
}
```


Listing 3.3: Zeek: example entry from conn.log in JSON format

Zeek's architecture is divided into two components: (*i*) the *event engine* that reduces the monitored network stream to discrete events that express high-level activity such as "new connection", and (*ii*) the *policy script interpreter* that interprets handlers for these events written in a domain-specific language (DSL) called *zeek script* (formerly *bro script*).

The original publication from 1999 [Pax99] focuses primarily on the design aspects of Zeek and its embedded script language as well as a discussion about the potential attacks on network monitors. While a specific evaluation section is not present, Paxson describes the application-level processing of the four protocols Zeek supported at that time, namely Finger, FTP, Portmapper, and Telnet. This already showed Zeek's support for simple rule-based alerting as these can be expressed via zeek script.

Since then, Zeek has evolved significantly both directly from active development by International Computer Science Institute (ICSI) members and Corelight[5] employees, as well as the largely community-driven package ecosystem[6]. At the time of this writing, Zeek is highly capable for deployments in both IT and (to some extent) OT environments and can detect several potential attacks by default, e. g., *Heartbleed*. The aforementioned package registry offers a large number of drop-in packages that require little to no customization and can perform various task from intrusion detection to performance monitoring. Examples for this include a framework to detect tactics, techniques, and proceduress (TTPs) from MITRE ATT&CK[7] (see: Section 2.2.3) or SSL/TLS fingerprinting via JA3[8]. As a consequence, Zeek is established as a popular open-source network monitor with usage in several high-profile companies and organizations.

This large script ecosystem combined with the growing set of protocols scripts shipped by Zeek as part of the distribution, enables security experts to leverage both complex rule-based alerting as well as sophisticated policy-based alerting algorithms. Due to

[5] see: https://corelight.com
[6] located at: https://packages.zeek.org
[7] see: https://packages.zeek.org/packages/view/cd5997f2-9348-11eb-81e7-0a598146b5c6
[8] see: https://packages.zeek.org/packages/view/cebd1c8c-9348-11eb-81e7-0a598146b5c6

the event-based architecture, all information extracted by third-party packages can be consumed in custom event handlers for higher-level detection logic. While some packages like the aforementioned JA3 are a perfect fit for rule-based alerting (on their fingerprints), the events generated for each TTPs from MITRE ATT&CK can be used as their own alerts or combined and correlated in Zeek to obtain higher-order meta alerts. Listing 3.4 shows a comparatively simple event handler for SMB that can detect transfer of executable files through the protocol.

```
1  @load base/frameworks/notice
2  @load base/frameworks/files
3  @load base/protocols/smb
4
5  module MyModule;
6
7  export {
8      redef enum Notice::Type += {
9          MyModule::SMB_Executable_File_Transfer
10     };
11 }
12
13 event file_state_remove(f: fa_file) {
14     if ( f$source != "SMB" || ! f?$info || ! f$info?$mime_type ||
       f$info$mime_type != "application/x-dosexec" || ! f?$conns) {
15         return;
16     }
17     for (id in f$conns) {
18         local conn = f$conns[id];
19         NOTICE([
20             $note = SMB_Executable_File_Transfer,
21             $uid = f$info$fuid,
22             $msg = "Executable file transfer via SMB",
23             $ts = f$info$ts,
24             $id = id,
25             $suppress_for = 1hrs,
26             $identifier = cat(id$orig_h, id$resp_h, id$resp_p)
27         ]);
28     }
29 }
```

Listing 3.4: Zeek example script to detect transfer of executable files via SMB

Overall, Zeek remains a valuable asset in any security infrastructure that can be adapted to many different scenarios. Although, rule- and signature-based detection alone is not sufficient to detect APT activity, Zeek can be leveraged as part of the APT detection process for tasks ranging from security monitoring to security analytics. We also chose Zeek as the underlying NMS to implement our approach for passive TLS decryption via cooperative endhosts.

3.4.2 Anomaly Detection

Anomaly detection has increasingly been used for alert generation in APT contexts. Compared to traditional policy-based detection, these approaches autonomously learn models of normality based of benign system activity and are able to mark anomalous events, which deviate from this model, as alerts. This is especially interesting for APT campaigns as they are known to leverage and misuse existing benign tools that oftentimes do not trigger alerts in traditional systems.

Friedberg et al. [Fri+15] propose a novel approach to detect APT activity based on anomaly detection of system events via log line analysis. The approach is an extension of [SFF14] and is meant to complement existing alerting solutions such as IDSs/IPSs and provides an additional source of alerts that can be analyzed in a higher-level security system such as a security information and event management (SIEM). The evaluation is performed on three datasets that were generated by a semi-synthetic data generation approach combining synthetic data and productive log data.

The approach is based on a model of normality that is derived from log files and is learned without any prior knowledge about their syntax or semantics. This system model is defined as a four-tuple: $M = \{\mathbb{P}, \mathbb{C}, \mathbb{H}, \mathbb{R}\}$, where

- $\mathbb{P}$ are *search patterns* used to classify information from ingested log lines;
- $\mathbb{C}$ are *event classes* classifying log lines according to the search patterns;
- $\mathbb{H}$ are *hypotheses* describing logical implications between log lines; and
- $\mathbb{R}$ are *rules* that represent proven hypotheses (proven by a hypothesis holding across extended periods of evaluation time).

The algorithms consists of four consecutive phases that are executed iteratively: (*i*) during *log-event extraction* single log atoms (e. g., a line in line-based logging or one XML-element) are timestamped and emitted as a log event, (*ii*) during *fingerprint generation* each log atom is vectorized to a fingerprint using the search patterns P from the system model, (*iii*) during *fingerprint classification* the previously generated fingerprint is classified to $[0, |\mathbb{C}|]$ event classes and emitted as an event $E^C =< t, C >$ that consists of the timestamp of the log event t and the event class C, and (*iv*) during *rule evaluation* the relations between log events are examined, hypotheses are updated, and potentially upgraded to rules, and an alert is raised if a rule is violated. All phases besides the first also contain a refinement Component that generates and updates $\mathbb{P}$, $\mathbb{C}$, and $\mathbb{H}/\mathbb{R}$ respectively.

The authors evaluate their approach on three semi-synthetic datasets generated by an approach previously presented in [Sko+14]. Scripts were used to simulate users on virtual machines which were monitored. The behavior of these virtual users is based on an analysis of a similar productive system over three months. Two of the three datasets only contain clean systems without any anomalies, while the third dataset is divided into a clean training phase of seven hours and an attack phase consisting of two time slots that each contain four malicious actions executed consecutively. The attack scenarios were selected to approximate APT behavior with attacks like data exfiltration or disabling of logging facilities to remain undetected.

The evaluation covers: (*i*) the configuration parameters of the approach, (*ii*) the detection performance, and (*iii*) applicability of the approach in industrial control systems (ICSs)/cyber physical systems (CPSs). In the first experiment, the two clean datasets are used to iteratively find stable configuration parameters based on six metrics including number of patterns, event classes, and rules and FPR. As the dataset is clean, all detected anomalies can be classified as a false positive. The results indicate that a stable configuration is highly dependent on the target network, which is expected. Nonetheless, the evaluation can be interpreted as a guideline how to tune the approach to a custom network. The second experiment focuses on the general detection performance of the approach using the default classification metrics FPR and TPR but based on single rule evaluations. The results indicate solid detection performance for both attack scenarios. In the first case, the TPR ranges between 40% and 80% (depending on the configuration and with one outlier) with a FPR

between 2% and 9%. The approach performed better in the second scenario with a consistent TPR around 80% and a FPR between 2% and 8%. In the third experiment, the authors briefly discuss the applicability of their approach in an ICS scenario. Based on a one-hour dataset obtained from an Austrian utility provider that consists of a firewall, a switch, and a supervisory control and data acquisition (SCADA) system, the approach was able to detect an injected anomaly with a TPR of 100% and a FPR of "nearly zero". This suggests that the approach is very well suited to ICS environments, which is however not surprising as OT system act much more consistently that IT systems and are thus relatively easy to capture in an anomaly detection model.

Based on the evaluation results, *R1: Accuracy* is marked as partially fulfilled. While the raw TPR and FPR values are good, only single anomalies were injected which is not representative of complex attacks. As each anomaly is related to a learned or predefined rule that should offer some limited context about the attack, *R2: Explainability* is partially met. *R3: Low overhead* is also fulfilled as the approach is based on log files which usually impose a tolerable overhead. However, *R4: Scalability* is marked as unknown, as no explicit runtime or scalability experiments were performed. Similarly, *R5: Security* cannot be assessed from the information provided in the publication. *R6: Privacy* is partially fulfilled as no explicit effort was taken to minimize the processed information obtained from coarse-grained log files. Lastly, *R7: Deploy- & Maintainability* is met as the approach relies on log files that are usually either available already or can be easily collected. Thus, the system should be relatively easy to deploy and maintain.

StreamSpot [MMA16] is a clustering-based approach for anomaly detection problems in heterogeneous streaming graphs. The approach is able to process multiple interleaved graphs in parallel with constant space and time requirements and achieves speeds of up to $100K$ edges per second while detecting anomalies in real-time. StreamSpot uses a novel similarity function based on *k-shinglings* to maintain updateable graph sketches that are clustered via a centroid-based scheme. The distance to the closest centroid is then used to score the updated graph and potentially report it as anomalous. While the problem area is not limited to APT detection, the authors explicitly chose it as their motivation and design their evaluation accordingly.

StreamSpot is evaluated in two ways using the same dataset of simulated system-call flow graphs of three scenarios containing a variety of benign activity as well as a drive-by download attack. First, the *static* evaluation analyzes StreamSpots overall detection performance to spot anomalous graphs in offline data and given unbounded memory. The results indicate that, given the optimal values for the parameters C (chunk length of the shingles) and K (number of medoids used in clustering), StreamSpot is outperforming the baseline of iForest [LTZ08] reliably. However, another experiment analyzes the impact of the percentage of the dataset that is used for training on the precision indicates that StreamSpot has problems with small training sets ($< 40\%$). In the second section, the authors investigate StreamSpots performance for its intended purpose, namely classification of *streaming* graphs. Their results indicate that the arrival of new graphs in the streaming process cause repeated drops in both accuracy and average precision. While the performance essentially "recovers" after each dip, the results raise questions on the real-world applicability of the approach, as decisions often need to be made in real-time and without ground truth available. The remaining results indicate that the approach is resource efficient (in both time and memory) as claimed in the introduction. Overall, the evaluation

presents StreamSpot as a decent anomaly detector for streaming graphs with some caveats regarding its accuracy.

The detection performance evaluation has some caveats that impede StreamSpot's accuracy, thus only partially fulfilling *R1: Accuracy*. *R2: Explainability* is not met as the resulting anomalies do not offer any additional context about the attack and thus need heavy manual investigation. The overhead of system-call capture is not investigated in the paper and thus *R3: Low overhead* is marked as unknown. *R4: Scalability* is fulfilled as StreamSpot should be able to scale to large numbers of hosts with its bounded resource requirements. Based on the data available in the publication neither *R5: Security* nor *R6: Privacy* can be assessed. Lastly, *R7: Deploy- & Maintainability* is not met as reliable capture and transfer of the high-volume system-call data remains an open challenge that is not explicitly addressed in this contribution.

PrioTracker [Liu+18] is a priority-based approach for attack causality analysis on system provenance data. It uses a concept of event rareness based on a reference model of typical host behavior derived from collected data of 150 machines in a real-world enterprise network. The approach supports human threat analysts in their forensic analysis of APT attack campaigns by highlighting abnormal causal relations in the provenance graph during incident response.

PrioTracker was designed with three core requirements: accuracy (i. e., capture the malicious system events), time effectiveness (i. e., find the maximum amount of abnormal behavior under time constraints), and runtime efficiency (i. e., introduce as little overhead as possible to the underlying causality tracker). According to the authors, the focus on timeliness is new and has not been considered in related work. A threat model based on previous work is also given in form of a TCB consisting of the kernel mechanisms, storage database, and the causality tracker. However, the adversary is expected to have full knowledge of "normal" activities on the host in the network, such that intentional manipulation is possible. The aforementioned reference model was derived from provenance data from 150 hosts (54 Linux- and 96 Windows-based) in an enterprise network. As the model is highly dependent on being homogeneous, the overall model is divided into three submodels corresponding to three internal departments in the observed network. Events were normalized and abstracted into Backus-Naur form (BNF) to enable matches between different hosts independent from system details. Furthermore, events were counted in time windows to prevent poisoning from repeated malicious activities. Each submodel then contains a *reference score* for each system event as the accumulative occurrence of this event across all hosts in the submodel.

The core algorithm in Priotracker is based on the algorithm to track event causality by King and Chen [KC03; KC05]. The algorithm (as shown in Figure 3.3 iteratively builds a dependency graph starting from an intrusion point *se* until either no more dependencies can be explored, or the time limit is reached (line 4). Instead of a regular queue, PrioTracker leverages a priority queue based on a priority score describing the event's abnormality. This ensures that the abnormal events from the set of currently available events are processed first. Newly discovered events are inserted into the queue with their priority (line 9) that is calculated from three factors: (*i*) the *event rareness* (how the event compares to the reference model), (*ii*) *fanout* (the amount of direct dependencies the event generated), and *dataflow termination* (a penalty for events that touch read-only or write-only files). The optimal weights used to calculate the priority were obtained via the Hill Climbing algorithm [RN03].

```
1:  procedure PRIOTRACK(se, T_limit)
2:      PQ ← ∅
3:      PQ.INSERT(se, Priority(se))
4:      while !PQ.ISEMPTY() and T_analysis < T_limit do
5:          e ← PQ.DEQUEUE()
6:          G ← G ∪ e
7:          E ← COMPUTEDEPS(e)
8:          for ∀e' ∈ E do
9:              PQ.INSERT(e', Priority(e'))
10:         end for
11:     end while
12:     return G
13: end procedure
```

FIGURE 3.3: PrioTracker: Dependency tracking algorithm [Liu+18]

The authors evaluate PrioTracker on a dataset captured from the same 150 hosts that were used to derive the reference model across one week. It contains 1TB of 2.5 billion system events. The authors then injected eight attacks that are representative of common APT activity into the dataset and compared PrioTracker with a reimplementation of [KC03] as a baseline. The extensive results indicate that PrioTracker is able to outperform the baseline implementation in both accuracy and timeliness. Largest improvements are seen in the timeliness category, where PrioTracker significantly reduces the time to find *critical events* of an attack. The evaluation also covers three attacks in more detail to explore how event prioritization speeds up the analysis. The case studies show how various properties of the attacks such as non-standard paths for common programs or abnormal transitions between processes result in high priorities and thus faster discovery of the related dependencies in PrioTracker. Runtime overhead in the analysis phase is also measured as low as 0.95μs on average which is mostly attributed to efficient query caching.

The results presented in the evaluation indicate a good detection performance for the injected attacks thus checking *R1: Accuracy*. The highlighted anomalous edges and nodes support SOC analysts in their investigations, however, the visualizations does not show larger overlapping context between incidents. *R2: Explainability* is thus marked as partially fulfilled. The low overhead stated in the publication is related to query time during the investigation and does not include the provenance capture on live systems. This leaves *R3: Low overhead* also at partially met. *R4: Scalability* is fulfilled given the scale and scope of the evaluation test data set. Based on the data available in the publication neither *R5: Security* nor *R6: Privacy* can be assessed. Lastly, like other system-call/data provenance-based systems, *R7: Deploy- & Maintainability* is not met as the deployment and maintenance of network-wide provenance capture remain difficult.

Rubio et al. [Rub+18] describe a graph-based approach to model, detect, and track APT activity in OT environments. The proposed algorithm is based on their previous work in [RAL17] and uses opinion dynamics to capture APT behavior and persistence. The authors simulate and evaluate their algorithm in Matlab based on insights obtained from a literature study of past APTs.

The approach is based on a graph model that represents an industrial control network with IT, OT, and firewall components. The graph $G(V, E)$ can be divided into the

respective subgraphs $G(V_{IT}, E_{IT})$ and $G(V_{OT}, E_{OT})$ with firewall nodes V_{FW} connecting both sections. To match real-world deployments, IT network sections are modeled according to a small-world network distribution, while OT network sections follow a power-law distribution of type $y \propto x^{-\alpha}$. Next, the authors describe how APT campaigns can be modeled based on the graph model and the commonly used kill chain models (see: Section 2.2). Each campaign is represented by a sequence of attack stages $attackSet_{APT}$ where each stage is one of nine available attack stages (*initialIntrusion*, *compromise*, *targetedLateralMovement*, *controlLateralMovement*, *spreadLateralMovement*, *exfiltration*, *destruction*, and *idle*). Each attack stage is defined on $G(V, E)$, e. g., randomLateralMovement is defined as malware delivery from a node n_i to a random neighboring node n_j (from $neighbours(n_i)$). Noteably this approach requires relatively concrete knowledge of the APT in question, i. e., which attack stages are used and in what order. The sequence $attackSet_{APT}$ can then be used as input for the detection algorithm.

The detection algorithm is a multi-agent algorithm that uses agents that detect local malicious activity as opinions (e. g., via anomaly detection), that are iteratively adjusted by the weighted averages of other agents in the network. The weight factor in this update is based on the distance between nodes as well as a notion of node criticality. This results in clusters of similar opinions once an attack is detected by a single agent. The algorithm iteratively calculates the opinion scores of all agents as well as aggregated *delta indicators* for the entire network as well as the IT and OT sub-networks that describe the overall anomaly score for this network (section). This enables analysts to both assess the general state of the network as well as to trace APT campaigns as they traverse the network as the opinion clusters move with the adversary.

The authors evaluate their approach in a small simulated industrial control network implemented in Matlab. The network is composed of three IT and OT components as well as a connecting firewall. The experiment Consists of a simulated Stuxnet attack that is modeled as a sequence of nine attack stages. Meanwhile, the opinions are iteratively updated and captured to evaluate how the attack influences them. The results indicate that the approach is able to successfully trace APT activity as the opinion across the nodes moves with the campaign progressing. Furthermore, the delta indicators reflect the attack progressing from the IT section to the OT section.

R1: Accuracy is marked as partially fulfilled based on the decent simulation-based evaluation results. As the opinions indicate which node was affected, but do not offer extended context about the attack, *R2: Explainability* is also partially checked. Unfortunately, as the authors also acknowledge, the simulated nature of the evaluation limits the comparison with most of the requirements. As such, *R3: Low overhead*, *R4: Scalability*, *R5: Security*, and *R6: Privacy* are marked as unknown. Lastly, the limited amount of data required for the approach marks *R7: Deploy- & Maintainability* as partially fulfilled.

UNICORN [Han+20] is an anomaly detection-based approach to detect APT activity in provenance data. The supervised machine learning (ML) approach summarizes growing provenance graphs as fixed-size sketches to enable efficient computation of graph statistics and ultimately learn a model of benign system execution. Anomalies to that model are then classified as APT attacks during testing. UNICORN operates in a four-step pipeline (as shown in Figure 3.4): (*i*) a streaming provenance graph is captured during live operation, (*ii*) UNICORN periodically summarizes graph

features of the streaming graph into variable-sized histograms, (*iii*) the variable-sized histograms are converted to fixed-size sketches via Histosketch [Yan+17], and (*iv*) UNICORN clusters the obtained sketches into a normality model that is later used for classification.

While the approach is not directly limited to APT detection, the authors made several design decisions to address the specific challenges that APT campaigns incur on detection such as the timely dispersion. Specifically, UNICORN uses exponential weight decay [Kli04] for its histogram elements to gradually phase out older data points while retaining causality. This enables the approach to focus on new host activity while still being able to track stealthy intrusion attempts that stay idle for extended periods of time after the initial infection.

The authors evaluate UNICORN with three distinct datasets: (*i*) the approach is compared to StreamSpot [MMA16] via their dataset comprising six scenarios. This evaluation demonstrates UNICORN's overall performance as an anomaly detector without specific APT focus. The results show that it offers precision and accuracy improvements compared to StreamSpot. (*ii*) UNICORN is specifically evaluated for APT detection performance via three APT trace sets captured as part of the fifth adversarial engagement from the Defense Advanced Research Projects Agency (DARPA) Transparent Computing program [DAR20]. Here, the approach reaches near-perfect F1 scores with 0.99, 0.99, and 1.0, respectively. (*iii*) the authors create two simulated APT supply-chain attack datasets to specifically evaluate runtime performance in terms of processing speed and resource efficiency. In this last section, the authors demonstrate that UNICORN's runtime if not influences by its parametrization and that is exhibits low CPU utilization and memory usage. Overall, the evaluation results indicate that UNICORN is a capable anomaly detector that can be used to detect APT attacks on host-level via data provenance.

Given the good detection results, *R1: Accuracy* is fulfilled. However, this missing context in the output alarm is the reason that *R2: Explainability* is not checked. *R3: Low overhead* and *R4: Scalability* are met as the runtime analysis indicates that resource usage scales with the number of patterns in the graph. Furthermore, a single host running UNICORN should be able to process provenance data from numerous hosts in the network. *R5: Security* and *R6: Privacy* cannot be assessed from the data in the publication. *R7: Deploy- & Maintainability* is not fulfilled due to the problems with reliable data capture as with other systems based on system-call/data provenance graphs.

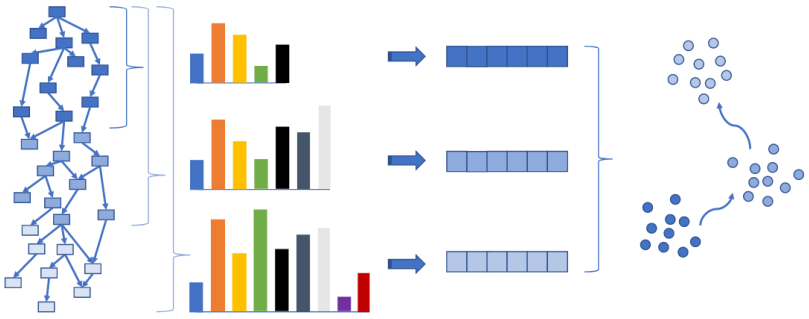


FIGURE 3.4: UNICORN: Pipeline overview [Han+20]

Pagoda [Xie+20] is an approach to detect APT activity by performing anomaly detection on provenance graphs. Compared to the authors' previous work PIDAS [Xie+16], Pagoda calculates anomaly scores not only for single paths in the provenance graph but for the whole graph as well, thus increasing detection performance.

Pagoda aims to fulfill three design goals that are based on requirements for general intrusion detection[9]: detection accuracy, real-time capabilities, and low overhead. The authors specifically mention that overhead encompasses three areas: disk space to build and store provenance data, memory overhead required by the detection mechanism, and the performance overhead the provenance collection imposes on normal system activity. From an architectural standpoint, Pagoda consists of six modules:

1. *Provenance collection*: in the reference implementation, provenance data is collected via PASS [Mun+06], but other frameworks are explicitly supported;
2. *Provenance pruning*: this component removes unneeded data from the provenance stream by filtering it. This encompasses (*i*) short-lived entities such as temporary files or pipe and (*ii*) all entities attributes besides name and relationship as Pagoda only requires this information;
3. *Provenance storage and maintenance*: captured provenance data and relations are stored in four non-volatile Redis [ST22] databases for quick access. Many-to-one relations (e. g., multiple files accessed by a single process) are merged into a collection in Redis to lower the storage overhead and improve detection performance;
4. *Rule building and deduplication*: rules that describe normal system activity in the provenance data are also stored in Redis. This component maintains this database and builds new rules based on observed dependency relationships in provenance graphs of benign behavior;
5. *Detection process*: this component is responsible for detecting intrusions and generating alarms. The algorithm first calculates anomaly scores for all paths starting on a head node (node without incoming edges) based on edge anomaly scores and raises an alert if a predefined path threshold is exceeded. Otherwise, the anomaly score of the whole graph is calculated, by taking a weighted average across all paths, and an alarm is raised if a predefined graph threshold is crossed; and
6. *Forensic analysis*: the last module supports forensic queries after an attack has been detected. An analyst can use the four provenance databases to execute forward and backward queries from known point(s) of intrusion to gain more insights about the adversarial actions that took place.

The authors evaluate Pagoda on real provenance traces from related [Ma+17] and own prior work [Xie+13; Xie+16]. The scenarios include multiple benign scenarios with varying system load such as benchmarks, kernel compilation, and regular developer activity as well as multiple attack scenarios that feature real vulnerabilities there were exploited to perform an attack. Overall, the dataset consists of 3 613 provenance traces across seventeen applications/scenarios.

The comprehensive evaluation features six sections that analyze (*i*) detection rate and FPR, (*ii*) both path and graph thresholds, (*iii*) detection time, (*iv*) query time, and (*vi*) overhead. In the experiments, Pagoda is compared with a baseline publication from 1998 [HFS98] as well as the authors' previous work PIDAS [Xie+16]. The

[9] In fact, two of those goals are directly reflected in the requirements section of this thesis (Section 3.1)

Requirement	Friedberg et al. [Fri+15]	StreamSpot [MMA16]	PrioTracker [Liu+18]	Rubio et al. [Rub+18]	UNICORN [Han+20]	Pagoda [Xie+20]
R1: Accuracy	○	○	✓	○	✓	✓
R2: Explainability	○	✗	○	○	✗	✗
R3: Low overhead	✓	◌	○	◌	✓	✓
R4: Scalability	◌	✓	✓	◌	✓	✓
R5: Security	◌	◌	◌	◌	◌	◌
R6: Privacy	○	◌	◌	◌	◌	◌
R7: Deploy- & Maintainability	✓	✗	✗	○	✗	✗

TABLE 3.2: Requirement Comparison: Anomaly Detection
✓ := fully met ○ := partially met ✗ := not met ◌ := unknown/neutral

results indicate that Pagoda consistently outperforms the competitors in at least detection performance or FPR (or both). While the exact performance differs across the scenarios, the approach produces solid detection rates roughly ranging between 0.5 and 1.0. In the second experiment, the authors evaluate the impact of different (path and graph) thresholds on detection performance and ultimately recommend a path threshold of >= 0.3 and a graph threshold of 0.5. The next two experiments evaluate detection and query time, respectively. The results indicate that again Pagoda consistently improves upon the compared approaches. However, comparisons with a Redis-based PIDAS indicate that some parts of the improvements can be attributed to the memory database. The last experiment Considers the various aspects of overhead mentioned in Pagoda's design goals. The results indicate that Pagoda improves upon PIDAS in size of the rule database while also requiring less memory during detection. The employed provenance pruning reduces the size of captured provenance compared to the raw size and a brief section mentions that runtime overhead of provenance capture is comparatively low with 0.69–6.30%.

The good results for both detection rate and FPR mark *R1: Accuracy* as fulfilled. However, as the alarms do not offer further context about the detected attack, *R2: Explainability* is not met. The remainder of the evaluation confirms that both *R3: Low overhead* and *R4: Scalability* are fulfilled as the overhead (including provenance capture) is comparatively low and the system scales well with available resources. As with many other approaches in this taxonomy class, *R5: Security* and *R6: Privacy* cannot be assessed from the available data in the publication. Lastly, again similar to other system-call/data provenance-based approaches, *R7: Deploy- & Maintainability* is left unfulfilled as the provenance capture remain a challenge.

Summary Table 3.2 gives an overview about the requirements fulfilled by the approaches for alert-level anomaly detection discussed in this section. As the table shows, there is no solution that checks all our requirements. Additionally, there are two interesting facts to note: (*i*) no approach manages to achieve both *R1: Accuracy*

and *R2: Explainability*. This supports our hypothesis that complex anomaly detection models (that are able to achieve good detection performance) fail to provide accurate explanations for their results thus not effectively supporting human SOC analysts. (*ii*) both *R5: Security* and *R7: Deploy- & Maintainability* are not usually explicitly covered by the surveyed approaches. While this is somewhat expected as the systems do not directly interact with any hosts in the network and just ingest the collected monitoring data, the requirements remain important nonetheless and should be discussed at least briefly. To address the missing combination of *R1: Accuracy* and *R2: Explainability*, we developed an approach to restore explainability for graph-based anomaly detection system that we present in Section 5.2. Our concept is based on a commonly applied technique from explainable artificial intelligence (XAI), namely input permutation importance, and we evaluate is using UNICORN [Han+20] as a black box approach. UNICORN was chosen as it checks some important requirements such as *R1: Accuracy*, *R3: Low overhead*, and *R4: Scalability* and is openly available.

3.5 Alert Correlation

This section discusses relevant approaches for alert correlation, i. e., approaches that aim to identify clusters of related alerts that share common properties or belong to the same underlying attack. They are not APT specific and usable for "general-purpose" intrusion detection. Techniques include clustering via various similarity metrics or merging of similar alerts, e. g., based on overlapping source or destination IP addresses. While they can offer decent results for simple and lightly complex attacks, alert correlation approaches usually fail to correctly identify alerts from APT campaigns due to their low volume and overall stealthiness. As this area of research is large and not directly related to APT detection, this section only highlights a selection of four relevant approaches.

Roschke et al. [RCM11] propose an algorithm for alert correlation based on attack graphs (AGs) that aims to reveal attack scenarios from a set of alerts. The publication describes the underlying formal model and a prototype implementation that is evaluated on a dataset captured at the authors' university. The algorithm consists of five steps: (*i*) preparation, (*ii*) alert mapping, (*iii*) aggregation, (*iv*) alert dependency graph generation, and (*v*) search for related alert subsets. During preparation, the required information including system and network information, alert classifications and the AG for the network is loaded. In the prototype implementation, MulVAL [OGA05] is used to generate the AG.

The remaining steps are then based on the formal model, that basically describes how alerts are matched to specific steps in the AG. The authors define an AG as a set of vertices and edges: $AG = (V, E)$. Each vertex is a tuple of $v = (im, h, r)$ where im describes the impact, h the host, and r the referenced vulnerability of the vertex. Based on that, a mapping function with different match modes $\Phi(a, v)$ can be defined that map alerts to vertices. The authors define five match modes that are later evaluated: `cvesrcdst`, `cvedst`, `cve`, `srcdst`, and `dst` that each use different information from the alert to perform the match. For the following steps, the model defines how alerts are aggregated based on similarity and temporal closeness, how dependencies of alerts are identified, and the strategy to search for related alert as part of scenario detection.

The authors evaluate their algorithm on a dataset captured at their university network consisting of alerts in Intrusion Detection Message Exchange Format (IDMEF) [SEC22]

format generated by *Snort* [Cis22]. To obtain an attack trace, a dedicated subnet with a Snort instance and vulnerable hosts was set up and subsequently attacked. The obtained traces were then injected into the larger, clean trace of the whole university to obtain a realistic dataset. The first experiment analyzes the impact of match modes on alert filtering performance. The results indicate that the match modes that included CVE information perform best with a filtering rate of 99.98% while `srcdst` and `dst` achieve 95.78% and 95.58% respectively. In the second experiment the impact of the temporal aggregation threshold is investigated. The results indicate that a threshold of 60 seconds helps to aggregate 67.86% of alerts without loosing accuracy in the best case.

The evaluation results indicate a good detection performance for the simulated multi-step attack especially if CVE-based matching is used. However, *R1: Accuracy* is only partially fulfilled as details of the multi-step attack are missing and results for the other matching modes are less convincing. *R2: Explainability* is also partially met as the correlation is based on AGs which offer some additional context although the evaluation does not cover this and only measures the number of correctly correlated alerts. *R3: Low overhead* is checked as the approach relies on low-overhead alert properties only. *R4: Scalability* cannot be reliably assessed from the information given as the authors only briefly mention an underlying platform based on "multi-core hardware". *R5: Security* and *R6: Privacy* cannot be evaluated from the data in the publication. Lastly, *R7: Deploy- & Maintainability* is partially checked as the approach relies on common alert data only that should be available in enterprise settings. However, the requirement is not specifically measured or discussed.

Daneshgar and Abbaspour [FA16] present an approach for fuzzy online alert clustering that is based on a causal relationship criterion represented by a *correlation strength matrix*. The correlation of the continuous alert stream is handled via a *fading model* compared to the usual sliding window-based approaches. This has the benefit that window size does not have to be defined and alert can be correlated across larger timespans as the lifetime of a pattern is dynamically derived from its activity. The authors evaluate their approach on two well-known intrusion detection datasets: DARPA2000 [MIT00] and ISCXIDS2012 [Shi+12].

The proposed alert correlation framework consists of two modules: the *online incremental fuzzy clustering module* and the offline *fuzzy frequent pattern mining module*. The first modules is responsible for clustering the online alert stream into fuzzy attack events. For this it leverages the usual similarity features of source and destination IP addresses as well as the correlation strength matrix, a data structure that stores the relationships among alert types. The second module ingests the attack events generated by the first module and applies fuzzy pattern mining, more specifically an adaption of frequent inter transaction item set mining [Chi+11], to continuously refine the matrix. This allows the approach to gradually adapt to environmental changes. While the statistical analysis required to update the correlation strength matrix is computationally intensive, the combination of online and offline processing enables the overall framework to process the alert stream in real-time.

The authors evaluate their approach in three experiments: (i) the online module processes the DARPA2000 [MIT00] dataset with an empty correlation strength matrix to estimate the detection performance without any statistical knowledge (neither predefined nor learned). Their results indicate that the both scenarios (LLDOS1.0 and

LLDOS2.0.2) are mostly clustered as attack events based on IP addresses and timestamps alone but miss a crucial attack step as the attacker uses IP spoofing as part of the attack which causes the approach to not include the alert in the attack event. (*ii*) the same scenarios from DARPA2000 are processed again with the initialized correlation strength matrix that was learned from the first experiment. As expected, the detection now improves and includes all relevant alerts as the statistical information obtained in the first experiment informs the online clustering process through the matrix. (*iii*) in the last experiment, the approach processes the ISCXIDS2012 dataset [Shi+12] again in to passes. The results here are unclear, as they appear identical across both runs, but the author mention the statistical information again improving the attack event. Overall, the evaluation is decent but lacks details about the improvements made through the correlation strength matrix and does not cover other relevant details about the approach such as the fading function.

While the evaluation shows good results, it has three problems thus not fulfilling *R1: Accuracy*: (*i*) it uses DARPA2000 for the first two experiments, an out of time dataset that is no longer representative of contemporary attacks, (*ii*) the second experiment using the new ISCXIDS2012 dataset assumes knowledge from the DARPA2000 experiment in the correlation strength matrix, and (*iii*) the results for the newer dataset are unclear with two identical tables that and conflicting results mentioned in the text. *R2: Explainability* is partially met as the statistically learned data offers some context about correlated attack events. *R3: Low overhead* is also partially checked as the approach only relies on few alert features but requires potentially significant computations in the pattern mining module. *R4: Scalability*, *R5: Security*, and *R6: Privacy* cannot be assessed from the information available in the publication. Lastly, *R7: Deploy- & Maintainability* is also partially fulfilled as the required alert information should be already present in network-based alerts in enterprise environments and the approach is designed in a modular fashion.

Graph-based Alert Correlation (GAC) [HF18; HF19] is an approach to reveal multi-stage attacks in large seemingly unrelated alert sets. The process consists of three consecutive phases: (*i*) *alerts* are *clustered* based on the connection tuples (IP addresses and ports). (*ii*) during *context supplementation* the communication patterns inside each cluster are identified and used to identify the potential scenario the cluster represents. (*iii*) matching clusters are *interconnected* to reconstruct the underlying multi-stage attack.

Graph-based Alert Correlation (GAC) leverages a variation of community clustering, namely the *clique percolation method* (*CPM*) [DPV05], to cluster the alerts in the first phase. For this algorithm, the choice of the clique size parameter k is essential as it defines the minimum size of a cluster. For GAC this could either mean, that small clusters are found, which are too specific and contain only a part of the attack or too broad clusters that contain alerts from multiple unrelated attacks. In the second step, the communication patterns (One-to-One, One-to-Many, Many-to-One, and Many-to-Many) are identified for each cluster and depending on which attributes from the alerts overlap, a scenario label is assigned, e. g., *worm*, *DDoS*, etc. These labels help to characterize the attack in the cluster and are also used to connect multiple attacks in the last phase. Here the Jaccard distance is used to calculate overlap between attacker and victims host sets to ultimately reveal a multi-stage attack.

The authors evaluate GAC in two ways: (*i*) synthetic datasets are used to separately analyze the first two phases of GAC. With the available ground truth, the authors

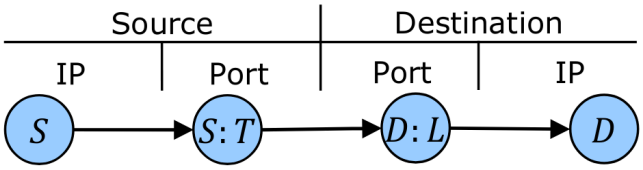


FIGURE 3.5: Haas et al.: Four nodes generated for an alert in the CSG [HWF19]

systematically measure the performance of alert clustering and context supplementation in the presence of multiple overlapping attacks and false positives via a blurring parameter β. The results indicate that k should be chosen from the interval $[24, 80]$ to achieve best results in both alert filtering and attack isolation. Furthermore, GAC is highly resilient against false positives when identifying the respective scenario of a cluster and can maintain a TPR of 1 until the blurring parameter β becomes larger than 0.3. (*ii*) GAC is evaluated on real-world data obtained from DShield [SAN22] without available ground truth. However, the results show that GAC retains high confidence for a majority of cluster labels (higher than 0.9 on average). Combined with the results from the synthetic evaluation, this indicates a near-complete TPR of 1. Additionally, a subset of labels was investigated manually and confirmed.

Contrary to the good evaluation results *R1: Accuracy* is not met as the evaluation is heavily based on large attack volumes that are not common in APT scenarios. Furthermore, the evaluation of the scenario reconstruction based on the real-world dataset is insufficient for our purposes as DShield does not offer APT datasets and the hashed labels make complete reconstruction impossible. *R2: Explainability* is also not fulfilled as the scenario reconstruction and multi-stage linkage offer very limited context to the threat hunter, which is also too coarse-grained to aid detection in APT scenarios. Due to the low resource usage for the approach itself and the lightweight alert format that it requires (only connection tuples), *R3: Low overhead* and *R4: Scalability* are fulfilled. *R5: Security* and *R6: Privacy* cannot be assessed from the available data. Lastly, *R7: Deploy- & Maintainability* is partially fulfilled as the required alert features (connection tuple) are available in enterprise contexts although this was not specifically evaluated.

Haas et al. [HWF19] propose an alert correlation algorithm to find and identify attack scenarios based on network motifs [Mil+02]. Alert clusters are transformed into communication structure graphs (CSGs) from which fixed-size motif signatures are calculated. These signatures are about 1% in size of the original alerts and can thus be efficiently exchanged in a collaborative setting.

The approach consists of four steps: (*i*) alert clustering/correlation, (*ii*) graph generation from alert clusters, (*iii*) motif signature calculation, and (*iv*) motif signature comparison. The first step is out of scope for this publication as related approaches can be used. Instead, the authors focus on correlation of clustered alerts. The CSG consists of four nodes for each alert. Figure 3.5 shows the four nodes that are generated for a single alert that describes communication from source host S with source port T to destination host D and destination port L as well as the edges between them. After all alerts have been added and the CSG is finalized, its motif signature can be calculated. This is done by enumerating all subgraphs of size n of the CSG, deriving the respective n-motifs for them, and counting the occurrences for all motifs. The resulting fixed-size vector is then converted to a Z-score (c.f. [Mil+02]) to enable comparison of multiple graphs independent of their absolute size. This Z-score is then

called the *motif signature* of the alert cluster. In the last step, these motif signatures are compared to find similar clusters. The authors chose a similarity function that interprets both signatures as vectors in a multidimensional space and uses the angle ϕ between them to be independent of the vector length. The resulting function, as given in Equation (3.1) is used on conjunction with a threshold τ to decide if two signatures (and the alert clusters they were generated from) belong to the same attack scenario.

$$sim = \frac{cos^{-1}(\phi)}{\pi} \tag{3.1}$$

This similarity based on motif signatures can then be used for three use cases: (*i*) signature-based classification, (*ii*) unsupervised clustering to dynamically learn new attack scenarios, and (*iii*) derivation of reference scenarios from an attack cluster. For signature-based classification the signature of an unknown cluster is compared with references signatures from a representative scenario, e. g., worm or DDoS. If the signatures match according to the similarity function, the cluster is labeled with the scenario. To learn new scenarios, hierarchical clustering with the threshold τ can be used to dynamically group similar clusters. The resulting clusters then represent new attack scenarios with unknown labels. The last application scenario takes unknown attack scenarios from the previous use case and derives a reference signature from the cluster. This is achieved by selecting the signature that has the highest similarity to all other signatures in the cluster.

The authors evaluate their approach on both synthetic alert sets and real-world alert data from DShield [SAN22]. In the first experiment, the *intra-class similarity* (i. e., the similarity between attack clusters from the same scenario) and *inter-class similarity* (i. e., the similarity between attack clusters from different scenarios) are used to evaluate if the approach fulfills the core classification requirement. The results indicate that the intra-class similarity consistently exceeds the inter-class similarity independently of the number of hosts in the scenario. This suggests that the classification is sound for the synthetic data. In the second experiment, the influence of τ on the number of learned scenarios is examined. The authors predefine six core scenarios in this experiment and show how τ impacts the accuracy of the hierarchical clustering. Next, the real-world dataset from DShield is classified according to the six reference scenarios of which signatures were obtained from synthetic data. The results indicate that 76%–96% of attacks could be labeled with one reference scenario depending on the choice of τ with DDoS as the predominant attack scenario. The last experiment evaluates the unsupervised learning of new scenarios from the remaining real-world data.

While the evaluation results indicate a good detection performance, the evaluated scenarios like distributed denial of service (DDoS) and worm spreading are highly untypical for APT campaigns and thus the obtained performance cannot be assumed for their detection. *R1: Accuracy* is thus not checked. However, the additional context through the motif signatures adds some value for the SOC analyst, especially as the signatures can also be exchanged efficiently. As a result *R2: Explainability* is partially fulfilled. Due to the lightweight nature of the signatures, *R3: Low overhead* and *R4: Scalability* are fully met, as motif processing can also be distributed throughout the network if required. *R5: Security* and *R6: Privacy* cannot be assessed from the information in the publication. Lastly, *R7: Deploy- & Maintainability* is partially fulfilled

Requirement	Roschke et al. [RCM11]	Daneshgar/Abbaspour [FA16]	GAC [HF18; HF19]	Haas et al. [HWF19]
R1: Accuracy	○	✗	✗	✗
R2: Explainability	○	○	✗	○
R3: Low overhead	✓	○	✓	✓
R4: Scalability	◌	◌	✓	✓
R5: Security	◌	◌	◌	◌
R6: Privacy	◌	◌	◌	◌
R7: Deploy- & Maintainability	○	○	○	○

TABLE 3.3: Requirement Comparison: Alert Correlation.
✓ := fully met ○ := partially met ✗ := not met ◌ := unknown/neutral

as the approach is designed to be deployed distributed in the network although this was not explicitly evaluated.

Summary Table 3.3 gives an overview about the requirements fulfilled by the approaches for alert correlation discussed in this section. As the table shows, there is no solution that checks all our requirements. More specifically, the statement from the beginning of this section is reinforced as not a single approach manages to fully meet *R1: Accuracy*—arguably the most important requirement. Similar with the anomaly detection approaches discussed in Section 3.4.2, we also see a lack of focus on *R5: Security* and *R6: Privacy*. One the positive side, we can see a partial focus on *R3: Low overhead* and *R4: Scalability* which is helpful when running these approaches in large enterprise settings as common in APT scenarios. Although the missing accuracy makes it unfeasible to only rely on alert correlation approaches, they can still be valuable as part of the overall APT detection pipeline to reduce alert volume by pre-clustering alerts before further processing takes place. We also employ this technique in our approach for APT campaign reconstruction based on Kill Chain State Machines (KCSMs) that we describe in Section 5.3.

3.6 APT Stage Detection

This section discusses relevant approaches for APT stage detection, that aim to identify malicious activity from single, selected APT stages as described in the established kill chain models (see: Section 2.2). While research has shown that defense against a single APT stage is not sufficient (as once potentially suggested in the original IKC [HCA11]), reliable detection of single stages is still important as a layered defense-in-depth strategy, e. g., by incorporating stage-specific alerts into higher-level detection approaches such as APT campaign detection and reconstruction discussed in the next

section. Approaches in this taxonomy class are further structured into three subclasses that match three selected APT stages that are recognized as highly important by literature [Uss+16]: *reconnaissance*, *command & control* (*C2*), and *lateral movement*. For detailed descriptions of these (and other) APT stages, we refer the reader to the UKC [Pol21] as covered in Section 2.2.4.

3.6.1 Reconnaissance

Reconnaissance is usually the earliest APT stage that can be detected in the target network. In this stage, attackers aim to obtain information about the target network they can later use to breach it and achieve the initial compromise. While the stage also contains steps completely outside of the target network such as research about the target infrastructure from publicly available data sources, attacker might also probe selected externally-accessible hosts to detect running software versions or other valuable information. As such, such services (especially related to authentication) like telnet, RDP, or SSH need to be monitored. This is also an active area of research with proposals approaches to detect brute-forcing activity on such services, e. g., by Najadabadi et al. [Naj+15], Hynek et al. [Hyn+20] or Hossain el al. [Hos+20]. However, these approaches usually aim to differentiate between failed login attempts or benign users and systematic brute-force login attempts. As such they cannot be leveraged directly for detection of APT reconnaissance activity. Javed and Paxson [JP13] define their threat model as "stealthy, distributed" brute-force attacks. While this is similar to tactics employed by APT adversaries, the authors also name failed logins of legitimate users as the baseline. As such, these approaches are not further evaluated with regard to our overall requirements that we formulated in Section 3.1. However, the information contained in the login attempts is still valuable for reconnaissance detection and can provide behavioral indicators that can help to categorize IP addresses. Owens and Matthews [OM08] as well as Abdou et al. [ABvO16] both propose several features and metrics for such characterization of brute-force login activity. We leverage these metrics and two novel ones in our contribution for brute-force characterization that we present in Section 4.2. The resulting clusters and metric results can be used to, e. g., filter IP addresses that engage in automated high-volume scanning and focus on the remaining login attempts.

Yamada et al. [Yam+15] describe an approach for real-time detection of remote access trojan/remote administration tool (RAT)-based APT reconnaissance activity in internal networks. This is highly relevant as APTs often leverage legitimate remote administration tools once a network zone has been breached. While the authors define this as reconnaissance, other models such as the UKC [Pol21] describe such activity as pivoting. The approach analyzes a limited set of features of the communication such as duration of the connection or presence of encryption to distinguish between legitimate user traffic and malicious reconnaissance.

Before the approach is proposed, the authors discuss behavior of both malicious RAT tools and SMB-based remote administration tools like *PsExec*. This results in a set of five features for RAT and a sequence of three actions for SMB-based tools that are identified to indicate malicious activity. Additionally, the publication discusses a combined attack that leverages RAT for probing and access from the Internet to an internal host and a SMB-based tool to move inside the target network as common with APTs. Unfortunately, the detection approach is only discussed briefly and not described in great detail.

The authors evaluate a prototype implementation of their approach in two ways: (*i*) they execute each discussed RAT in combination with one of four SMB-based remote management tools. The results indicate that the approach is able to detect 99.29% of the possible combinations only missing a single one. (*ii*) the prototype is deployed in an internal network of an unnamed organization where it records and analyzes 50–60 million TCP connections over a 22-hour period. The results indicate that the approach is robust against false positives in real-world settings as none are recorded during this experiment. Overall, the evaluation shows promising results. However, the scope and lack of detail in the description of the experiments leaves some doubts about the approach's effectiveness.

Given the limited evaluation, especially the lack of detailed description of the first experiment, *R1: Accuracy* is not fulfilled. *R2: Explainability* is marked as partially met, as the set of features is limited and thus should allow the SOC analyst to obtain some context about detected incidents. Due to missing information in the evaluation, the requirements *R3: Low overhead*, *R4: Scalability*, *R5: Security*, and *R6: Privacy* cannot be assessed. Lastly, *R7: Deploy- & Maintainability* is partially checked as the sensor is network-based and seem to not require complicated deployments.

Haas et al. [HWF20] describe a correlation algorithm for distributed port scan campaigns based on ten key features that describe both attacker intention and the tools used to perform the scan. These features include typical connection attributes such as *source* and *destination ports* and *IP version* as well as per-IP statistics such as *number of target hosts*, *attempts per host/port combination* or geolocation data of the source IP address, among others. The authors also give a similarity definition for each feature ranging from 0 (no match) to 1 (exact match).

The correlation algorithm first calculates the features for every IP that has been observed in any incoming connection to obtain a feature-based fingerprint. Next, a *weighted average* over the similarity of the features is used to determine if two scanning IP addresses are colluding in a joint scan campaign. To find these collaborative scanners in large alert sets, the approach uses *unweighted pair group method with arithmetic mean* (*UP-GMA*) [SS62], a variant of hierarchical clustering. This has the advantage that the number of clusters does not need to be known in advance and the most similar scan attempts are clustered first as the algorithm operates in a bottom-up manner. To adapt to different scenarios, the algorithm is configurable by three core parameters: (*i*) the number of ports X that are considered *few* and not *many*, (*ii*) the minimum number of scan probes an IP address must perform before it is considered part of a larger campaign ϵ, and (*iii*) the similarity threshold for clustering t.

The authors evaluate their scan correlation algorithm on a dataset captured by the MAWILab project [Fon+10]. More specifically, the trace file consists of IP, and TCP/UDP headers with randomized last bytes in all IP addresses from a 15-minute window captured by MAWILab on 2019-05-05. While this is the only dataset used for evaluation, the authors simulate smaller networks by filtering traffic by subnet to reduce visibility. The resulting scenarios are *Internet backbone* (baseline/all traffic), *Internet Service Provider* (netmask 133.242.0.0/16), and *enterprise network* (netmask 133.242.179.0/24). For traffic analysis the Zeek NMS [Pax99] is used as it can detect incomplete TCP handshakes that are most likely the result of a port scan. The evaluation is split into two parts: (*i*) a parameter study is conducted to determine their optimal values for the dataset. The authors arrive at $X = 10$, $0 < \epsilon < 10$ and $t = 0.15$, although the justification is based on limited observations. (*ii*) The campaigns obtained by the

approach are evaluated on an exemplary basis. Out of 1955 total detected campaigns from the dataset, the authors analyze two in greater detail across the three scenario visibility levels. Their results indicate that large parts of the features stay stable across the scenarios and thus allow a solid clustering even with restricted visibility. While the absolute numbers of target hosts vary (due to the scans not being limited to the respective visibility level), key characteristics like the choice of source ports remain stable. Overall, the approach seems to provide some good indicators of colluding scanner activity, although the improvement over some previous work is not always clearly visible.

The presented results are promising but somewhat limited and thus *R1: Accuracy* is partially fulfilled. The same is true for *R2: Explainability* as the correlation does help to find potential collusion but offers limited additional context about the detected clusters. *R3: Low overhead* is also partially met as the required data can be captured on network-level without much overhead on the endhosts, although overhead for the approach itself is not measured. The requirements *R4: Scalability*, and *R5: Security* cannot be assessed based on the data in the publication. *R6: Privacy* is partially met, as the required data for the approach is limited but not explicit privacy considerations are included. Similarly, *R7: Deploy- & Maintainability* is checked as the needed data can be easily captured on network-level.

3.6.2 Command & Control

Command & control (C2/C&C) is another important stage in APT campaigns as it solidifies the attackers' foothold in the network by establishing outbound connections to servers under their control. This allows attackers to obtain access to the compromised machines (e. g., via tunneled or reverse shells) and thus to expand their reach by subsequently compromising additional machines. As enterprise environments are usually tightly controlled, attacker leverage common application protocols like HTTP/HTTPS to perform C2/C&C communication as exotic or custom protocols are highly likely to get filtered at the network edge. As a result, most approaches in this section also focus on detecting C2/C&C communication based on HTTP/HTTPS.

DUMONT [SR11] aims to detect covert outbound HTTP communication by learning profiles of normal HTTPS requests of users of the monitored network. By deriving the profile from both structure and content of the outbound connections, DUMONT detects anomalies from the normality model and tags them as tunnels or other covert channels.

The approach is based on seventeen features that can be categorized in four semantic groups: *length features*, *structural features*, *entropy features*, and *temporal features*. DUMONT then uses a *One-Class support vector machine* (*SVM*) with *Gaussian kernels* [Mül+01] to learn a model of normality for the different features extracted from HTTP requests. The individual detectors for the separate features are combined in hierarchical layers to enable cross-feature detection of anomalous requests that are deemed normal in single-feature detection but exhibit anomalous patterns across multiple features. DUMONT's training process consists of two core steps: (*i*) during *model selection* two core hyperparameters (the width of the Gaussian kernel and the "softness" of the One-class SVM) are optimized for each feature detector. The selection process is driven by the FPR as well as the number of support vectors in the respective model. (*ii*) in the *automatic calibration* phase, the model of each detector is further improved by selecting an optimal soft margin radius of the SVM. This process favors a

low FPR over a high detection rate as false positives of the individual detectors accrue across the hierarchical detection layers.

The authors evaluate DUMONT on two datasets: (*i*) a benign dataset obtained via an opt-in HTTP proxy server at their institute. This dataset consists of 182 996 requests from six different users across 90 days of capture. (*ii*) a malicious dataset obtained from executing 2 765 malicious executables and PDF documents as well as two open-source tools to establish covert communication that were executed in a controlled environment resembling a typical enterprise environment. This dataset includes 12 899 HTTP requests across 695 sessions.

These two datasets are used to evaluate the detection performance of DUMONT in isolation and in comparison with Webtab [BP04], an approach that aims to detect covert communication via filters, trained rules, and thresholds. The results indicate that DUMONT is able to reliably detect covert communication via HTTP with (average) TPRs between 89.3% and 100% depending on the class of malicious traffic (tunnels, web backdoors, and malware) while also maintaining an (average) FPR of 0.35%. Additionally, the authors show that DUMONT is largely resistant to user agents masquerading (a technique often employed by malware to evade detection). DUMONT's (average) TPR drops only slightly from 89.3% to 82.0% while Webtab's TPR drastically decreases from 100% to 3.7%.

Given the good results especially in the presence of masquerading, *R1: Accuracy* is met. However, as the underlying model is opaque and only tags anomalies without additional context, *R2: Explainability* is not fulfilled. *R3: Low overhead* is partially checked, as the system operates on HTTP data that can be captured without imposing a lot of overhead on the hosts. However, no explicit measurements were taken for the calculation. Which leaves *R4: Scalability* as unknown. *R5: Security* and *R6: Privacy* cannot be assessed from the given information. Lastly, *R7: Deploy- & Maintainability* is marked as partially fulfilled as HTTP capture can usually be facilitated in enterprise context without much additional work.

DECANTeR [Bor+17] is an approach to *passively fingerprint* applications that communicate via the HTTP protocol. These fingerprints are then used to detect unknown malware, backdoor/C2/C&C communication or data exfiltration. To address the problems of incomplete datasets and changing host configurations, the approach is able to adapt the learned fingerprints over time.

As typical with learning-based approaches, DECANTeR runs in two different modes: *training* (to passively learn application fingerprints) and *testing* (to detect anomalous fingerprints). The training phase consists of two steps: (*i*) during *labeling* HTTP requests are clustered according to their `User-Agent` field. This is done to efficiently identify benign applications that use this header to identify themselves. Next, these clusters are labeled either *background* (communication not directly influenced by user input that is often highly predictable) and *browser* (unpredictable communication from interactive user input). This labeling is achieved by a novel algorithm to generate a *Referrer Graph* to effectively reconstruct the standard browser behavior of cascading HTTP requests from dynamic web content. (*ii*) during *fingerprint generation* the previously established clusters are tagged with a fingerprint based on a selection of features out of six total features (*Host*, *Constant Header Fields*, *Average Size*, *User Agent*, *Language*, and *Outgoing Information*). After the training phase is finished, DECANTeR

has learned a set of fingerprints of benign applications running on the host in the monitored time span.

In testing mode, DECANTeR continues to generate fingerprints as in training but also adds a *detection* step to identify anomalous fingerprints. This is done by computing similarities between the newly found fingerprints F_{test} and the known fingerprints from training F_{train}. If any new fingerprint is distinct from F_{train} DECANTeR estimates if the new fingerprint is the result of a software update to update the knowledge base. If this is not the case, the approach raises an alert if the new fingerprint either *resembles a browser user agent* (as this is common malware behavior) or contains an amount of *Outgoing Information* is above a configured threshold σ (which may indicate a data exfiltration attempt). The authors evaluate DECANTeR on four datasets that represent different use cases: (*i*) a *user dataset* (*UD*) obtained by capturing HTTP traffic of university researchers during working hours, (*ii*) an *organization dataset* (*OD*) comprised of outgoing HTTP traffic of a large international organization captured via Zeek [Pax99], (*iii*) a dataset based on communication generated by *Data Exfiltration Malware* (*DEM*) samples that were analyzed using Cuckoo Sandbox [Cuc22], and (*iv*) a *ransomware* (*RAN*) dataset consisting of 287 pcaps from ShieldFS [Con+16] that contain communication from ransomware samples that produced at least 100 bytes of HTTP traffic.

At the start of the evaluation the authors briefly discuss the trade-offs when choosing the two configuration parameters of DECANTeR: the trigger threshold σ and the aggregation time during testing t. Based on the UD dataset, the authors describe how both parameters influence FPR and response time and set $t = 10$ minutes and $\sigma = 1000$ bytes for the following experiments. The evaluation of the detection performance on the UD, RAN, and DEM datasets shows satisfactory results with regard to both TPR (76–88%) and FPR (0–2%) respectively. Additionally, a comparison with DUMONT [SR11] revealed that DECANTeR consistently outperforms it on a per-request basis (instead of DECANTeR's regular per-fingerprint mode) for UD, RAN, and DEM. A discussion of the obtained results from the OD dataset, potential *evasion techniques*, future improvement opportunities, and DECANTeR capabilities to detect *data exfiltration* attempts concludes the paper.

R1: Accuracy is fulfilled based on the good results (especially compared to DUMONT [SR11]). Additionally, DECANTeR is also usable to detect data exfiltration further improving it's usefulness. However, as the system also does not directly offer additional context about the detected anomalies, *R2: Explainability* is not met similarly to DUMONT. *R3: Low overhead* is partially met as the approach is also based on HTTP fingerprinting (thus incurring low overhead on hosts) but no explicit experiments have been conducted in this direction. *R4: Scalability*, *R5: Security*, and *R6: Privacy* cannot be assessed from the given information. Lastly, *R7: Deploy- & Maintainability* is partially checked due to the availability of HTTP data in enterprise scenarios.

Hawk-Eye [AM21] is an approach to extract and detect APT C2/C&C domains from pcap files. The classification is done via a random forest that uses *semantic*, *contextual*, and *hybrid* features of domain names and associated data such as `WHOIS` and `NS`. The authors selected those features based on an extensive literature study spanning 63 APT campaigns of the last 13 years.

The result of this study is a three-phase APT model that focuses on C2/C&C channel establishment and usage throughout the campaign. In the first phase, *prepare C2/C&C*

infrastructure, the adversaries set up a complex network of domains and fall-back options depending on the strategic properties of the target such as geographical location or used services. Domain names and associated nameservers are then chosen that could belong to the target and thus be overlooked by SOC analysts, e. g., by *typosquatting* trusted domains (`0ffice.com` instead of `office.com`). The second phase, *prepare attack vectors*, describes activities that happen between C2/C&C establishment and usage, e. g., reconnaissance, weaponization, delivery, and exploitation, among others. The authors consider domains used exclusivity as part of this phase out of scope. In the third and final phase, *attack management*, the adversaries have partial control about the target network and carry out further actions. This phase effectively spans from first usage of the C2/C&C infrastructure to campaign end and thus can cover large timeframes. The authors observed key findings such as frequently changing IP addresses in the DNS `A` records, IP address reuse across or even in the same campaign and abuse of `TXT` records for data exfiltration or command distribution.

Based on the proposed model and underlying information extracted from literature, the authors compile a list of 70 metrics in total across three categories: (*i*) *semantic features* describe the properties of the fully-qualified domain name (FQDN) itself such as typosquatting/TLD squatting or usage of technical words to confuse analysts. This includes character features like number of vowels or consonants, the ratio between vowels and consonants, or simply the length of the FQDN and lexical features based on the tokenized words in the FQDN. (*ii*) *contextual features* describe the underlying DNS infrastructure that host the malicious domains. This includes feature like domain suffix, domain age, and registrar or nameserver reputation. (*iii*) *hybrid features* describe semantic features of additional entities (mainly domains and FQDNs) that were retrieved for the analysis of contextual features such as registrar or nameserver. In other words, the features from the first category are applied to entities queried in the second category. While 20 of these metrics have been used in previous work to identify malicious domain names, the remaining 50 metrics are newly proposed in this paper (mainly from the hybrid category).

Hawk-Eye consists of four modules: (*i*) the *parser* module extracts FQDNs from the DNS queries of a given pcap file and segments them into host, entity level domain, and suffix. These segments are then used to calculate semantic features and passed on further information retrieval. (*ii*) the *crawler* module obtains additional data such WHOIS and nameserver information as required for contextual and hybrid features and calculates them. To support past campaigns, Hawk-Eye is able to query both live information and historical data obtained from SecurityTrail's API [Sec22]. (*iii*) the *preprocessor* module normalizes the calculated metrics and encodes them for the ML-based classifier. (*iv*) the *classifier* module labels each FQDN based on the calculated features. While Hawk-Eye conceptually supports multiple types of classifiers, the authors chose a random forest.

Hawk-Eye is evaluated on a newly designed dataset [AM20] that consists of FQDNs from the analyzed APT campaigns as well as legitimate domains from Alexa and phishing FQDNs from PhishTank [Gro22]. The authors build three configurations of the dataset: HEAL (Hawk-Eye APT and Legitimate), HEAP (Hawk-Eye APT and Phishing), and HEALP (Hawk-Eye APT, Legitimate and Phishing). Additionally, two feature sets to use with Hawk-Eye are defined: *holistic*, that includes all 70 metrics as well as a *literature baseline* for comparison consisting of 20 features previously proposed in related work.

The authors evaluate Hawk-Eye in two main dimensions: detection performance and feature importance. Hawk-Eye is executed in the two feature configurations across all three dataset configurations. The results indicate a strong detection performance for the holistic feature set across all three dataset configurations with F1 scores of 98.53% (HEAL), 88.66% (HEAP), and 98.39% (HEALP) respectively. The biggest improvement over the literature baseline is obtained for the HEAP configuration where the baseline only reached 79.34%. The evaluation of the feature importance shows that the most important feature for most classification decisions is domain age (a feature that is already established for detection of malicious domains). However, several other features in the ten key features are newly proposed, thus explaining the improved performance for the holistic feature set. A discussion about details of the training and testing process as well as current limitations of the prototype rounds out the evaluation section. Overall, Hawk-Eye shows promising results for use cases in real-time APT C2/C&C detection and forensic work and the evaluation results support the proposed holistic feature set.

Due to the good results across all three configurations *R1: Accuracy* is checked. *R2: Explainability* is not fulfilled as the opaque model offers no context for the detected anomalies. As the system focuses on pcap files, which are relatively simple to capture, *R3: Low overhead* is partially met. *R4: Scalability*, *R5: Security*, and *R6: Privacy* cannot be assessed from the given information. Lastly, *R7: Deploy- & Maintainability* is fulfilled due to the simplicity of pcap capture in enterprise networks.

3.6.3 Lateral Movement

Lateral movement is another important stage of APT campaigns as it describes the compromise of additional machines. Every newly infected machine thus increases the attackers' foothold in the network and potentially grants access to new network resources like databases or CPSs. Detection of lateral movement is difficult as attackers usually leverage legitimate tooling like *PsExec* (in Windows-based networks) or SSH (in Unix-based networks). This has two key advantages as (*i*) no additional executables need to be delivered to the compromised machines and (*ii*) the resulting network traffic is less likely to be noticed as it blends in with legitimate traffic.

CloudSEC [Tia+19] is an approach to detect lateral movement in cloud or edge computing scenarios in real-time via evidence reasoning networks (ERNs) that are based on system vulnerabilities and environmental information. The resulting evidence chain helps the cloud operator to estimate the impact of the attack and supports mitigation.

CloudSEC consists of two components: the *EventTracker* and the *AlertCorrelator*. The EventTracker is deployed inside each VM or container running on the edge cloud host machine. One instance of the AlertCorrelator is deployed inside each cloud environment and correlates alerts generated by network-based IDSs in it. Both components feed information to the ERN instance for the respective cloud environment. Additionally, the authors describe a central management unit, which is used for information exchange and for evaluation. The ERN describes vulnerability correlations of one network and is a directed graph G described by the following five-tuple: $G = \{N, E, L, W, D\}$, where

- N is a set of vertices that describe vulnerability information;
- E is a set of edges that connect correlated vulnerability vertices;

- L is a set of logical expressions that describe the relationships expressed by the edges $e \in E$ as either `AND` or `OR`;
- W is a set of risk weights for each vulnerability vertex $n \in N$ describing the likelihood this vulnerability is successfully exploited; and
- D is a set of data structures associated to each vulnerability vertex $n \in N$ including evidence storage and pointers to parent vertices.

G is generated from the topology of the target network, its vulnerabilities (which can be detected by tools such as nmap [Ln22] or Nessus [Ten22]) and the predetermined set of risk weights W. The publication further illustrates this process with an example network.

The ERN can now be used for evidence chain reasoning. Given a list of evidence from an attack in the network, each piece of evidence can be mapped to a vertex in the ERN based on the vulnerability used and other auxiliary data such as the timestamp. After all evidence has been mapped, a breadth-first search (BFS) is executed to trace the path through the graph that represents the attack chain that generated the evidence. In addition to the regular evidence chain reasoning process, the authors also propose a *timing-independent* evidence chain reasoning method to account for incorrectly timestamped evidence.

The authors evaluate CloudSEC in two experiments using the DARPA2000 [MIT00] and Treasure Hunt [UC 04] datasets. In the first experiment, evidence chains were generated for the LLDOS1.0 and LLDOS2.0.2 campaigns from DARPA2000. For LLDOS1.0 all malicious events were captured, resulting in a *confidence* of 1.0. The evidence chain also matches the descriptions provided by the creators of the dataset. For LLDOS2.0.2 some events were missed by the sensor, thus reducing the confidence to 0.86. However, the chain was successfully reconstructed for the remaining events. The second experiment uses the Treasure Hunt dataset [UC 04] which was generated in a network with three subnets at UC Santa Barbara. According to the authors, the dataset is similar to edge cloud environments as there are a multitude of attacks present in the dataset. Here CloudSEC generated two evidence chains with confidences of 0.92 and 0.33, respectively. A brief performance evaluation indicates that CloudSEC is capable of real-time analysis.

While the obtained results indicate good performance, they have been generated from comparatively old datasets, thus only partially fulfilling *R1: Accuracy*. The ERNs offer some context about the detected activity but fail to concisely summarize the details. This leaves *R2: Explainability* also partially checked. *R3: Low overhead* and *R4: Scalability* are also both marked as partially met due to the brief performance evaluation that indicates overall low resource usage of the approach. *R5: Security* and *R6: Privacy* cannot be assessed from the available data in the publication. Lastly, *R7: Deploy- & Maintainability* is not met, as the EventTracker needs to be present on each host/VM significantly hindering simple enterprise deployments.

Amin et al. [Ami+21] leverage hidden Markov models (HMMs) to predict lateral movement and use the predictions to deploy decoy nodes to hinder the attacker's progress. If the attacker does not follow the predicted path, the approach suggests partially observable Monte-Carlo planning (POMCP) as a second layer of defense to assess available mitigation strategies.

The approach works in three stages: (*i*) *HMM parameters are trained* offline based on observations of past lateral movement attacks, that were detected prior (e. g., by means

of employing an IDS) and a CVE database with information about vulnerabilities. (*ii*) during online detection, an IDS generates alerts that are post-processed via an alert correlation module. These correlated alerts are then fed to the *HMM prediction module* to predict likely attack paths and attack probabilities based on Bayesian attack graphs (BAGs). (*iii*) once a lateral movement attack is detected and an attack path has been predicted, *decoy modules are deployed* and a defense policy assessment module aid the defender in luring the attacker towards the decoy nodes.

The HMM parameters are trained via the Baum-Welch method based on historical alert observation from lateral movement attacks. The algorithm starts with initializing the HMM parameters (A, B, Π) randomly. Next, the probability of transitioning from state i to state j at time t as well as the marginal probability over state j is calculated. Last, these two values are used to compute the converging and thus locally maximized HMM parameters. During online detection, alerts are first deduplicated and correlated to obtain attack chains via depth-first search (DFS). This alert log is then passed to the prediction unit which uses the trained HMM parameters as well as a preconfigured BAG to capture success probability of exploits. This BAG needs to be predefined by an administrator or another security professional. The module then calculates the attacker's most likely next step based on the information available.

The defense assignment aims to solve two core defender goals: (*i*) it quantifies the security state and attacker's progression through the network and (*ii*) it recommends the optimal defense action based on the current state and the defender's capabilities. The security state is modeled as an exploit dependency graph that contains the set of security conditions as well as the set of exploits and their pre- and postconditions. Additionally, a belief matrix is employed to represent the defender's partial knowledge about the security state and the attacker type. The defense algorithm based on POMCP [SV10] simulates available mitigation actions based on the current security state and an estimation of the attacker's options to recommend an optimal solution.

The authors evaluate their approach on two large-scale network simulations that resulted in 100–110 million security state respectively. MulVAL [OGA05] is used to generate the attack paths and ArcSight enterprise security manager (ESM) [Mic22] to extract the attack sequence from an alert set. The results indicate that the approach is able to generate a comparatively small set of likely online attacks paths that contains the path the attacker took. Furthermore, decoy nodes were effectively used to deter the attacker from the real goal and thus hinder the lateral movement.

The results obtained in the evaluation are mostly promising but slightly unclear to correctly judge as no clear definition of accuracy or precision is given. *R1: Accuracy* is thus marked as partially fulfilled. The forecast about potential next steps the attacker might take can support SOC analysts thus checking *R2: Explainability*. *R3: Low overhead* was not a focus of this work and is thus left as unknown. To address runtime efficiency, the authors mention offline precomputations of selected steps, but as not concrete scalability experiments were performed, *R4: Scalability* is marked as partially met. *R5: Security* and *R6: Privacy* cannot be assessed given the data in the publication. Lastly, *R7: Deploy- & Maintainability* is not fulfilled as the required data is very detailed and is non-trivial to obtain in enterprise contexts.

Summary Table 3.4 gives an overview about the requirements fulfilled by the approaches for detection of single APT stage discussed in this section. The table highlights, that there is no approach for either of the three selected APT stages, that

	Rec.		C2			LM	
Requirement	Yamada et al. [Yam+15]	Haas et al. [HWF20]	DUMONT [SR11]	DECANTeR [Bor+17]	Hawk-Eye [AM21]	CloudSEC [Tia+19]	Amin et al. [Ami+21]
R1: Accuracy	✗	○	✓	✓	✓	○	○
R2: Explainability	○	○	✗	✗	✗	○	✓
R3: Low overhead	◌	○	○	○	○	○	◌
R4: Scalability	◌	◌	◌	◌	◌	○	○
R5: Security	◌	◌	◌	◌	◌	◌	◌
R6: Privacy	◌	○	◌	◌	◌	◌	◌
R7: Deploy- & Maintainability	○	✓	○	○	✓	✗	○

TABLE 3.4: Requirement Comparison: APT Stage Detection.
✓ := fully met ○ := partially met ✗ := not met ◌ := unknown/neutral

checks all our requirements. Additionally, as established in Section 3.2, stage-specific approaches alone are not sufficient to detect complete APT campaigns anyway. Furthermore, the table shows three interesting facts to note: (*i*) only approaches aimed at revealing C2/C&C communication are able to achieve *R1: Accuracy* but consistently fail to meet *R2: Explainability*. This is a similar trend we also observed for alert-level anomaly detection as discussed in Section 3.4.2. (*ii*) contributions in this section put less focus on the performance characteristics as expressed by *R3: Low overhead* and *R4: Scalability*. (*iii*) there is no approach aimed at detecting lateral movement that fully checks *R7: Deploy- & Maintainability*, which is an important factor in enterprise environments. To address the shortcomings in the area of lateral movement detection, we propose an approach to reconstruct attacker lateral movement based on a generalized network model that we present in Section 5.1. Our proposal abstracts from vulnerability and topology information, that is also commonly employed by the approaches discussed here, into a formal graph-based model that we use to derive attacker models and to reconstruct likely infected hosts from an incomplete alert set. The abstraction helps with *R7: Deploy- & Maintainability* specifically as we abstract from the concrete alerts and available system information.

3.7 APT Campaign Detection and Reconstruction

This section discusses relevant approaches for APT campaign detection and reconstruction, i. e., approaches that aim to address the overall goals of this thesis. To achieve this, data from all lower levels is leveraged to reconstruct the overall APT campaign, e. g., by mapping alerts, alert clusters, and even events to a kill chain-based model (see: Section 2.2). The added contextual information is essential for SOC analysts & threat hunters and can be used for incident response and mitigation measures.

Bhatt et al. [BYG14] propose a Hadoop-based framework to identify APT stages and reconstruct the overall attack campaign based on semi-structured log files. The framework consists of five modules: (*i*) The *logging* module collects logs from IDSs (host- and network-based) as well as other security related logs and services such as web and mail servers. (*ii*) The *log management* module stores and preprocesses all logs in the HDFS and makes them accessible via queries. (*iii*) The *malware analysis* module investigates potential malware samples in a virtualized lab environment via code analysis or behavioral analysis. (*iv*) The *intelligence* module provides core functionalities of log correlation, kill chain stage detection, and APT scenario reconstruction. (*v*) The *control* module enables the administrators to configure and maintain the framework by deploying new rulesets or testing hypotheses via the intelligence module.

The proposed architecture describes a well-organized framework to detect APT attacks, however the description of some component is intentionally left out of scope (malware analysis) while others are only briefly explained. This hinders the comparison with the requirements established in Section 3.1. Additionally, the evaluation is shallow and only contains a single simulated APT campaign without detailed description of the simulated environment and the campaign itself.

R1: Accuracy cannot be assessed from the limited information given about the conducted experiment. The campaign reconstruction based on the IKC [HCA11] is not highly detailed but provides some useful context to SOC analysts. Thus, *R2: Explainability* is partially met. As the proposed system is based on log files, which can usually be obtained with limited overhead, *R3: Low overhead* is also partially fulfilled. *R4: Scalability* is also partially checked as Hadoop can be extended with additional cluster nodes, although this was not explicitly investigated in the publication. *R5: Security* and *R6: Privacy* cannot be assessed from the given information. Lastly, *R7: Deploy- & Maintainability* is partially met as the framework is divided into clearly separated components (thus simplifying deployment) and ingests log files that are likely already present in enterprise deployments.

HERCULE [Pei+16] is an approach to detect APT detection and scenario reconstruction based on log correlation. The authors model multi-stage intrusion detection as a *community discovery* problem common in social network analysis. The implementation correlates lightweight log entries per host on commodity hardware and discovers so-called *attack communities* embedded in the graph structures. The paper mentions six log sources that implementation is able to process (DNS, WFP connect, HTTP, process create, object access, and authentication) and 20 features that are extracted from them. The graph is built from log events as vertices while edges are added if one of 29 rules matches the properties of two vertices, e. g., same destination port or process name for both vertices. The resulting multidimensional graph is then searched for community cliques that are marked as attack communities.

The authors evaluate HERCULE on sixteen APT scenarios that they derived from previous reports and simulated on two victim and one attacker machine. While the scenarios are diverse and well selected, the limitation to three machines raises questions about the accuracy of the simulated attacks. This is especially important for lateral movement that the authors also identified as a key stage. Nonetheless, HERCULE shows decent results for accuracy and FPR for most simulated scenarios.

Due to these results, *R1: Accuracy* is marked as fulfilled. *R2: Explainability* is only partially checked, as HERCULE helps to reconstruct the attack story but still requires

significant human investigation as the authors also acknowledge. While the authors include a section about "logging overhead", the results are not sufficient to check *R3: Low overhead* so it is marked as partially met. *R4: Scalability* cannot be assessed as a no runtime performance evaluation is included. Similarly, *R5: Security* and *R6: Privacy* can not be assessed from the data given in the publication. Lastly, *R7: Deploy- & Maintainability* is partially fulfilled as the system processes lightweight log files that are likely already present in enterprise context.

SLEUTH [Hos+17] is a tag- and policy-based approach to detect and reconstruct complex attack campaigns (including APT scenarios) via provenance-based dependency graphs. The overall process is divided into four steps: (*i*) The *provenance data* is *captured* on the target systems. (*ii*) Platform-neutral *dependency graphs* are *constructed* where vertices represent processes, files, sockets, and other kernel-level objects and edges model audit events such as read, write, connect or bind. (*iii*) *Attack detection* is performed based on tags and pre-defined policies. For SLEUTH tags are coarse-grained labels that encode *trustworthiness* and *sensitivity* of the kernel objects. (*iv*) For alerts detected in the previous step, tags are once again leveraged to perform *root-cause and impact analysis*. The result are compact *scenario graphs* that describe the attack visually and aid the threat hunter in their analysis.

The attack detection uses policies to initialize and propagate tags through the provenance graph. Alerts are then raised if one of four core conditions are met: (*i*) untrusted code execution, (*ii*) modification by subjects with lower code trustworthiness tag, (*iii*) confidential data leak, and (*iv*) preparation of untrusted data for execution. This captures the expectation that attackers need to run untrusted code to compromise a machine and aim to exfiltrate data at some point during the campaign. The final bidirectional analysis aims to identify entry points and impacts of the attack and reconstructs the potential scenario by pruning, merging, and filtering the provenance graph.

The authors evaluate SLEUTH on eigth APT campaigns that were conducted by a professional red team as part of an unspecified adversarial engagement of the DARPA Transparent Computing program [DAR15]. From these campaigns, the approach correctly identified 174 out of 176 total malicious entities and was able to largely reconstruct the scenario graph. Additionally, the authors describe how custom policies can help to reduce false positives, e. g., for software update scenarios where untrusted code is read and potentially executed.

The positive results obtained for both detection performance and scenario reconstruction indicate that *R1: Accuracy* and *R2: Explainability* are fulfilled. The low runtime and memory use would check *R3: Low overhead* but as overhead on the capturing machines was not investigated it is marked as partially fulfilled. This also applies to *R4: Scalability* as the overhead is especially important when scaling the approach to multiple monitored hosts. *R5: Security* and *R6: Privacy* are left as unknown and cannot be further assessed from the data available. *R7: Deploy- & Maintainability* is not fulfilled, as the reliable setup and maintenance of provenance-based systems remains an open challenge for now.

HOLMES [Mil+19b] aims to detect ongoing APT campaigns via provenance analysis as they are happening and to provide high-level explanations of the different actions as a graph. It achieves this by generating alerts for kernel-level events that map to stages in the IKC [HCA11] and correlating them based on information flow

on provenance level to reconstruct the scenario. The resulting graphs are then pruned and prioritized for manual analysis by a human threat hunter based on a threat score. This score is assigned to each graph based on the APT stages it includes and their severity rating in CVSS [For15]. Similar to previous work, HOLMES assumes a thread model based on a TCB consisting of kernel mechanisms and the provenance capture system. Additionally, the system is assumed to start in a benign state, such that any attacks originate from outside the enterprise network.

To map low-level provenance activity to high-level kill chain stages, HOLMES uses an intermediate layer based on TTPs from Mitre ATT&CK (see: Section 2.2.3). TTPs are then defined as matches on the underlying directed provenance graph, the dependencies between events, and other prerequisites that can include specific folder, executable names, or other TTPs. Matched TTPs are then correlated to a High-level Scenario Graph (HSG) that represents them as nodes while information flow and causality relations are modeled as edges. Each HSG also carries a *threat score* that reflects how likely the graph contains an actual APT campaign. The score is based on subscores for all seven considered APT stages that are combined via a weighted product with configurable weights allowing the analyst to prioritize certain stages over others. Additionally, a detection engine computes the weighted sum of all HSGs obtained from a provenance graph and raises an alert if a threshold is surpassed.

The authors evaluate HOLMES on nine APT scenarios that were conducted as part of the third adversarial engagement from the DARPA Transparent Computing program [DAR18]. To measure the detection performance the threat score of benign and attack graphs are compared. The highest score for benign graphs was 338 while the lowest score of an attack graph was 608. This indicates that attacks were successfully separated from benign operations. Additionally, the authors derive a recommendation for the detection threshold based on F_1 score. Finally, further experiments positively rate HOLMES' runtime performance and memory usage with experiments running on a machine with moderate resources (150GB of RAM and an 8 core CPU at 2.5GHz).

R1: Accuracy and *R2: Explainability* are fulfilled as the threat score helps to accurately identify attack graphs that additionally also describe the attack stages well visually. While the evaluation showed low resource consumption for the detection and reconstruction parts, overhead on the capturing machines was not measured. This leaves *R3: Low overhead* and *R4: Scalability* as partially met. *R5: Security* and *R6: Privacy* are left as unknown as they have not been evaluated in the publication. As HOLMES also relies on provenance data and does not improve the deployment story of reliable capture, *R7: Deploy- & Maintainability* is also marked as not fulfilled.

NoDoze [Has+19] is an approach to rank provenance-based alerts by propagating anomaly scores throughout the corresponding dependency graphs based on *historical context*. Human analysts can then leverage the aggregated scores to better triage alerts for investigation. Additionally, NoDoze addresses the problem of graph explosion by using *behavioral execution partitioning* to reduce the size of the graphs to analyze while keeping causal dependencies of anomalous events.

The approach is deployed additionally to an existing provenance-based IDS and adds additional context to already generated alerts. The process is divided into four steps: (*i*) NoDoze assigns an anomaly score to all events in the dependency graph of newly detected alerts. This is based on historical frequencies of similar events occurring on all monitored hosts. (*ii*) the previously assigned anomaly scores are propagated

and aggregated along the graph via a novel network diffusion algorithm. This step simplifies the identification of related events later on. (*iii*) the dependency graph is assigned an aggregate anomaly score that can be used to triage the corresponding alert. (*iv*) NoDoze extracts the subgraph with the highest anomaly score from the dependency graph. Due to the propagation in step two, the extracted graph contains the most anomalous *sequence of events* that are highly likely to have caused the alert in question.

The authors evaluate NoDoze on a real-world dataset containing provenance data of 191 hosts across 5 days collected at NEC Labs America. In this timeframe, ten APT attacks as well as 40 malware samples were executed and a commercial IDS generated 364 total alerts. This dataset is then used to evaluate NoDoze in both its effectiveness and accuracy as well as its runtime overhead. The results indicate that the approach is able to reduce the size of the dependency graph by two orders of magnitude while retaining nearly all edges of the malicious actions. In the worst case NoDoze achieves 88% TPR for control dependency and 84% data dependency, thus capturing large parts of the attack scenario. Furthermore, the ranking based on aggregate anomaly scores significantly improves triaging as true positives are ranked higher. Based on estimations of 10–40 minutes for investigation of a single alert [Bro17; Cis18], the authors estimate that the triaging saves roughly 90 employee-hours in their five-day evaluation period. For their runtime evaluation the results indicate that NoDoze is able to respond in less than 200 seconds for 95% of alerts including time consumed by the underlying provenance tracker, this makes the approach feasible for near real-time deployments.

The results obtained for detection performance and the graph examples produced by NoDoze fulfill *R1: Accuracy* and *R2: Explainability*. The runtime evaluation shows some promising results in regard to response time but does not address computational and especially memory requirements directly, thus *R3: Low overhead* and *R4: Scalability* are only partially met. The add-one character of NoDoze also partially checks *R5: Security* as a failure or compromise of the system should not impact the remaining detection infrastructure. *R6: Privacy* cannot be assessed from the data available in the publication. Lastly, *R7: Deploy- & Maintainability* is not fulfilled as reliability and maintenance of the provenance capture and IDS are not directly addressed.

POIROT [Mil+19a] is an approach to correlate provenance data with cyber threat intelligence (CTI) to aid threat hunters in their search for APT activity. More concretely, the authors formalize the threat hunting problem as a *graph pattern matching* (*GPM*) *problem* of a *query graph* (i. e., a graph representation of a set of indicators of compromise (IoCs) that belong to the same APT attack) in a (larger) *provenance graph* from the monitored host. As the generalized GPM problem is NP-complete [DRT09], POIROT leverages an approximation function as well as a novel *similarity metric* based on (sub-)graph alignment to reduce the problem's complexity.

The authors describe the process of transforming intelligence reports into labeled, typed, and directed query graphs similar to the construction of provenance graphs. Entities (e. g., processes, sockets, files...) are represented as nodes while relationships between them (e. g., read, write, bind...) are modeled as directed edges between them. Additionally, the reports are generalized, e. g., by replacing concrete executable or file names by wildcards or substituting IP addresses with subnets or even just zones like "internal" and "external". For structured and semi-structured formats such as STIX [OAS21], OpenIOC [Man13] and MISP [MIS22] this process can be automated

with some manual fine-tuning while reports in natural language have to be converted by human analysts. With both graphs in place, POIROT tries to find subgraphs in the provenance graph that resemble the generalized query graph. In addition to GPM inherent complexity this is further complicated as a single edge in the query graph is more high-level than one in the provenance graph and thus likely represents a path in the provenance graph on its own. To address this, a novel graph pattern matching technique based on subgraph alignment is used. Once a match is found, POIROT calculates a *similarity score* between the query graph and the provenance subgraph and further raises an alert if this score is higher than a configured threshold.

The authors evaluate POIROT in four experiments to analyze its effectiveness in detecting real-world APT attacks. (*i*) provenance data of the third adversarial engagement from the DARPA Transparent Computing program [DAR18] is used. For the ten attacks that were investigated, the similarity scores vary between 0.54 and 1.00 (i. e., complete match of the query graph). This indicates that the approach offers good detection performance if the query graphs are precise enough. (*ii*) the authors select seven real-world incidents with publicly available threat descriptions in natural language and reproduce the attacks by executing their binaries in a controlled environment. The results indicate that POIROT is consistently able to find large parts of the query graphs with similarity scores between 0.62 and 1.00 even for malware mutations that were not part of the original threat reports. Three other analyzed tools only detect superficial properties such as exact hash signature or file name matches and, in some case, even fail to detect any malicious activity at all. (*iii*) the benign dataset from [DAR18] is used to analyze the FPR of POIROT on benign data and the optimal value for the threshold parameter. The results indicate that the approach is largely resistant to false positives as the largest similarity score for benign data is 0.16 and thus below the threshold of 0.33. (*iv*) the authors investigate runtime performance and show that POIROT is capable of real-time processing with search times below 1 minute and runtime overhead below 2%.

While the detection performance is evaluated well and shows promising results, *R1: Accuracy* is only partially met as the approach heavily relies on precise threat descriptions that are often unavailable a-priori for sophisticated APT attacks. *R2: Explainability* is fulfilled as even partially matched query graphs offer valuable context to SOC analysts and threat hunters. The runtime results and in particular the runtime overhead fulfill *R3: Low overhead* and *R4: Scalability*. *R5: Security* and *R6: Privacy* are marked as unknown and cannot be further assessed from the data available. Comparable to the other provenance-based approaches, *R7: Deploy- & Maintainability* is not checked as reliable and maintainable data capture is not addressed.

Summary Table 3.5 gives an overview about the requirements fulfilled by the approaches for APT campaign detection and reconstruction discussed in this section. As the table shows, there is no solution that checks all our requirements. Additionally, there are three things to note: (*i*) most approaches are able to fulfill *R1: Accuracy* and *R2: Explainability* while also partially addressing runtime and scalability concerns expressed by *R3: Low overhead* and *R4: Scalability*. Compared to anomaly detection-based approaches (as discussed in Section 3.4.2) the added explainability is somewhat expected as the campaign reconstruction offers significant context to SOC analysts. (*ii*) *R5: Security* and *R6: Privacy* are never met (with one exception) similar as with approaches from previous sections. (*iii*) *R7: Deploy- & Maintainability* remains an open challenge for approaches based on (whole-)system provenance as in previous

Requirement	Bhatt et al. [BYG14]	HERCULE [Pei+16]	SLEUTH [Hos+17]	HOLMES [Mil+19b]	NoDoze [Has+19]	POIROT [Mil+19a]
R1: Accuracy	◌	✓	✓	✓	✓	○
R2: Explainability	○	○	✓	✓	✓	✓
R3: Low overhead	○	○	○	○	○	✓
R4: Scalability	○	◌	○	○	○	✓
R5: Security	◌	◌	◌	◌	○	◌
R6: Privacy	◌	◌	◌	◌	◌	◌
R7: Deploy- & Maintainability	✓	○	✗	✗	✗	✗

TABLE 3.5: Requirement Comparison: APT Campaign Detection and Reconstruction
✓ := fully met ○ := partially met ✗ := not met ◌ := unknown/neutral

sections. To address the need for campaign reconstruction concepts that are able to both achieve high accuracy and remain simple to deploy & maintain, we developed a reconstruction approach based on KCSMs that we describe in Section 5.3. It is based on KCSMs that we derived from the UKC [Pol21] and does not rely on detailed (and thus hard to capture) system provenance data. To simplify deployment and integration with other tools in the detection stack, our approach is able to ingest alerts that are pre-tagged with a specific APT stage (as generated by the systems presented in Section 3.6) as well as untagged network-based alerts for which we generate stage mappings based on network direction.

3.8 Chapter Summary

This chapter introduced key requirements for comprehensive APT detection as well as a taxonomy of the research field. Literature from the five key areas of the taxonomy, namely *event-level monitoring*, *alert-level detection*, *alert correlation*, *APT stage detection*, and *APT campaign detection and reconstruction* was classified according to the taxonomy and compared with the proposed requirements. This evaluation highlighted that most approaches do not fully meet our stated requirements for APT detection.

Event-level monitoring encompasses approaches to obtain low-level visibility into both host- and network-activity. This visibility is the foundation for any higher-level detection and usually defined by technical challenges instead of open research questions. The presented approaches for (*i*) visibility into (TLS-)encrypted network traffic and (*ii*) data provenance generation closely relate to the main contributions of this thesis and were thus selected.

Alert-level detection approaches that aim to identify malicious behavior in the collected events and to generate corresponding alerts that describe this activity. Both signature- and policy-based detection are established concepts in this area, that are not sufficient for APT detection when used in isolation and thus not covered in detail in this thesis. Instead, relevant approaches based on anomaly detection were discussed

that specifically target APT activity. The comparison with our stated requirements revealed missing approaches that combine both *R1: Accuracy* and *R2: Explainability*.

Alert correlation describes approaches that link and cluster related alerts according to predefined rules or shared behavior. While this is useful to reduce overall alert volume and to reveal connected alerts for high-volume attacks, APTs usually produce fewer alerts thus limiting the effectiveness of traditional alert correlation. The comparison with our stated requirements confirmed our hypothesis that alert correlation alone is not sufficient to detect APT attacks as none of the discussed approaches fully checked *R1: Accuracy*.

APT stage detection approaches target single (or a limited set) of APT stages to maximize detection performance. The presented approaches were aimed at three key stages: reconnaissance, C2/C&C, and lateral movement. While detection of single stages alone is not sufficient to effectively mitigate APT campaigns, the high-confidence alerts for single stages can help as part of larger reconstruction efforts. The comparison with our stated requirements revealed that no approach achieves both *R1: Accuracy* and *R7: Deploy- & Maintainability*.

APT campaign detection and reconstruction describes approaches that aim to detect APT campaigns as a whole usually by correlating activity along a kill chain-based model such as the UKC [Pol21]. The generally good detection performance is mostly achieved by consuming large amounts of fine-grained system provenance data which poses challenges in enterprise environments. Consequently, the comparison with our stated requirements showed missing approaches that achieve both *R1: Accuracy* and *R7: Deploy- & Maintainability* similar as for stage-based APT detection approaches.

In summary, this chapter showed that APT detection is a complex process that relies on several different factors. Stage-specific approaches are too narrow to be used in isolation, but can help to identify key stages as part of the larger campaign. Existing approaches for APT detection can offer decent performance but often lack explainability (in the case of anomaly detection-based approaches) or require detailed provenance data for campaign reconstruction, which is hard to reliably capture and process. Thus, the following chapters of this thesis aim to advance the state-of-the-art for APT detection. Chapter 4 presents two approaches from the area of security monitoring to improve visibility on (*i*) the external threat landscape for brute-force login attempts and (*ii*) payloads of TLS encrypted communication inside the network. Chapter 5 presents three approaches in the area of APT detection: (*i*) a stage-specific detection algorithm for lateral movement primarily aimed at forensic scenarios, (*ii*) a campaign-based reconstruction approach based on a notion of kill chain state machines and (*iii*) a concept to restore explainability for blackbox APT detection approaches on graph data.

4 Approaches for Enhanced Security Monitoring

This chapter describes two main contributions that both aim to increase visibility for network security monitoring, i. e., information about activity (both benign and malicious) on network level. As briefly discussed in Section 3.2, this visibility on the lowest level is essential to obtain signs of advanced persistent threat (APT) activity in the first place. Higher-level detection approaches (as also proposed in the next chapter) can then correlate and otherwise process this information to reveal hidden interconnections. If approaches on this visibility level miss certain activity, it is "lost" throughout the detection process. The first contribution has been published at IFIP SEC 2022 [Wil+22] and received both *Best Student Paper* and *Best Paper* awards. It describes an approach for transparent TLS decryption in enterprise environments that leaves endhosts in control of the key material and thus allows for selective decryption. The approach offers substantial improvements for both security of the network as well as the privacy of users compared to the predominant approach of TLS interception and is described in Section 4.1. The second contribution was published at IEEE CNS 2020 [WF20b] and describes a collection of metrics that can be used to characterize and cluster brute-force login attempts. This way security operations center (SOC) analysts can gain an overview about the external threat landscape through this attack vector and obtain indicators to distinguish between "harmless" login attempts caused by automated scans and potential APT reconnaissance activity. This approach is described in Section 4.2. Both approaches improve security monitoring for increased internal and external visibility which is the foundation for higher-level APT detection and reconstruction algorithms, e. g., as presented in the next chapter.

4.1 Transparent TLS Decryption for Network Monitoring

This section presents an approach to restore access to cleartext communication payloads for network monitoring systems (NMSs) in enterprise environments. Regaining visibility into encrypted traffic of workstations is essential for APT detection as initial infections often occur via these machines. Overall, this contribution aims to answer research question **RQ1** as introduced in Section 1.1:

RQ1 *How can NMSs regain visibility into TLS-encrypted communication without actively intercepting the connections on a central entity and breaking end-to-end guarantees?*

We address this problem by deploying a daemon on all endhosts in the network that cooperatively forwards TLS key material to the NMS such that it can passively decrypt network traffic for further analysis. Compared to TLS-intercepting Man-in-the-Middle (MitM) proxies, our approach lowers the threat surface and can conceptually enable users to selectively retain privacy for some connections. This contribution is based on preliminary work from a Master's thesis [Sch20] supervised by the author of this

thesis. Furthermore, this section shares material with the corresponding conference publication [Wil+22] that revises and extends the approach developed in the Master's thesis.

4.1.1 Motivation and Objectives

Internet traffic is increasingly encrypted. Out of the 100 top non-Google websites (which account for about 25% of worldwide traffic according to Google), 97 default to HTTPS with the remaining three at least fully supporting it [Goo22]. This has tremendous privacy and security benefits for users as their data is protected against unwanted eavesdropping or modification by nosy ISPs, intelligence services, or cybercriminals. However, encrypted network traffic is troublesome in enterprise environments as it cannot be inspected by NMSs/intrusion detection systems (IDSs) anymore. As a result, these systems have to fall back to metadata-only analysis that is usually not sufficient to reveal indicators of APT activity. The predominant approach to address this are *TLS-intercepting proxy servers* that effectively MitM any TLS connection leaving the internal network. The proxies act as cryptographic endpoints of the connections between clients and servers. As a consequence, the proxy gains access to the cleartext payloads which can be forwarded to the existing detection systems to further analyze them. However, this approach has several drawbacks as outlined in Section 3.3.1.

To address this problem, this contribution aims to achieve the following *objective*: restore visibility into encrypted TLS traffic for a trusted NMS. In contrast to established practices like TLS interception and approaches from literature [Nay+15; Nay+17; Lee+19], the following *functional requirements* should be satisfied (in addition to the overall requirements formulated in Section 3.1):

1. **No interception**: TLS is a well-designed protocol to ensure end-to-end data confidentiality and integrity between two endpoints. Entities, that intercept TLS connections represent new and unwanted attack vectors that may weaken the security of the connection and break mutual authentication. Thus, a proper solution should not intercept the TLS connection.
2. **No protocol modification**: TLS is deployed widespread across a wide variety of devices and networks. Thus, changes on the protocol-level are unlikely to gain enough traction for widespread adoption. To strive for real-world usage, a solution should not require any changes to the existing TLS standards.
3. **No additional latency**: While security and network monitoring is an important topic in any enterprise network, it should not deter the user experience by introducing additional latency to the connection.
4. **End-to-end data integrity**: The NMS needs to break end-to-end confidentiality between client and server to perform analysis tasks on the cleartext. However, this process does not require integrity violation as application data only needs to be read by the NMS. Thus, a good solution should maintain end-to-end data integrity for inspected TLS connections if possible.
5. **Support for selective decryption**: Endpoints should be able selectively influence which connections are allowed to be decrypted by the NMS to let the user retain some privacy for sensitive or personal data. While high-security environments might require inspection of all network traffic, there are also scenarios in which complete visibility is not feasible, e. g., for legal or compliance reasons. A good solution should support such scenarios by enabling fine-tuned configuration

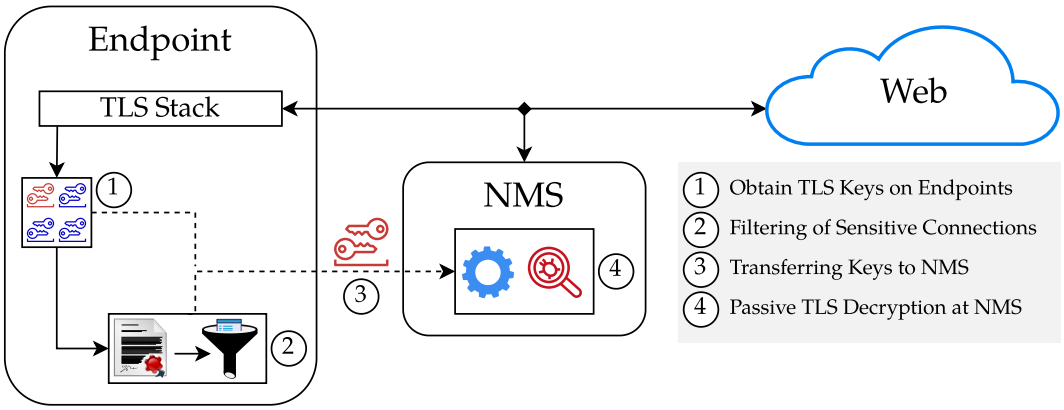


FIGURE 4.1: Overview: Passive TLS decryption via cooperative endpoints

about which connections are allowed to be decrypted and ideally enforce the decision on a conceptual level.

6. **Low maintenance effort**: Security monitoring is an essential service in an enterprise network and as such should require low maintenance for regular operations. This is both relevant for client machines and any middleboxes deployed in the network.

The remainder of the section is structured as follows: Section 4.1.2 gives a high-level overview about our approach for transparent TLS decryption via cooperative endhosts. Section 4.1.3 describes the four stages of our approach in more detail from key collection on endhosts to actual TLS decryption on the NMS. In Section 4.1.4 we discuss conceptual advantages of our approach compared to the current state-of-the-art and evaluate it on a dataset comprised of real-world web traffic including a comparison with the requirements formulated in Section 3.1.

4.1.2 Approach Overview

Figure 4.1 gives an overview about the complete process and components involved. First, the local daemon obtains TLS session keys. Second, it filters the key for sensitive connections according to policy. Third, they transfer the keys to the NMS. Fourth, the NMS matches the keys to connections and starts passive decryption and further analysis. Steps 1–3 are executed directly on the endpoint and do not require direct user interaction.

Keeping authority over keys on the endpoints and performing the decryption directly at the NMS offers increased flexibility. (*i*) The endpoint can enforce policy-based control about which connections are allowed to be decrypted by the NMS by withholding keys for sensitive connections. (*ii*) The NMS can opt to skip decryption for performance reasons, e. g., for video conferencing or streaming content. Both use-cases are not easily possible with current state-of-the-art MitM proxies. A concern compared to enterprise MitM deployments might be that visibility might be lost if an attacker should manage to disable the endpoint protection daemon thus disabling key transfer and preventing decryption. However, such an incident would already be detected and flagged even in legacy deployments, as the endpoint protection daemon is already monitored for liveness.

4.1.3 Approach Details

4.1.3.1 Obtain Key Material on Endhosts

In the first step, TLS keys need to be obtained from the client machines as they represent the cryptographic endpoint. This is done in the context of the privileged endpoint protection daemon that has access to the network stack. A typical TLS connection, uses multiple secrets as well as session keys for different purposes. The simplest solution would be to obtain the pre-master secret and forward it to the NMS with the client random as the connection identifier. The NMS would then perform key derivation as shown in Listing 2.1 and obtain all session keys (and initialization vectors (IVs) if an AEAD cipher suite is used) to decrypt the encrypted application data packets. However, this enables the NMS to derive all session keys including the integrity keys. This would allow a compromised NMS to forge packets on behalf of both endpoints and thus introduce an additional attack vector to the network. For non-AEAD ciphers, we can mitigate this by only forwarding encryption keys to the NMS and keeping the integrity keys used by the HMAC on the client. Unfortunately, AEAD ciphers require both encryption keys and IVs to decrypt the data even if no integrity checks need to be performed by the NMS. From now on, we use the term *keys* synonymously with either just the encryption keys (if the cipher suite allows for that) or encryption keys and associated IVs.

The keys can be obtained in multiple ways depending on operating system, cryptographic library, and application. The simplest way is the **SSL keylog interface** [MDN22] supported by widely used cryptographic libraries such as OpenSSL [TYH22] and NSS [NSS22]. If enabled, the libraries write TLS pre-master secrets and the corresponding client randoms to the file specified in an environment variable. While this approach is easy to deploy, it might introduce some problems: (*i*) The logfile is readable by the user, as the browser process is running with user permissions. (*ii*) Writing secrets to a file and reading them in the aftermath via an application adds additional latency. While this can be partially addressed by using a RAM disk, the NMS might receive the keys too late and the first few packets of a TLS connection cannot be decrypted.

The **in-kernel Transport Layer Security (kTLS)** interface [Wat16] offers another way to obtain the desired keys. kTLS was introduced to the Linux kernel in version 4.13 and moves the TLS packet handling to kernel space. At the time of writing only a limited subset of AEAD cipher suites is supported. To use it, an application opens a regular socket and performs a normal TLS setup via key exchange and derivation. Next, it configures the encryption key as well as IV for the socket via a `setsockopt` system call. From that point on, regular `send` and `receive` operations can be used with the kernel performing encryption and integrity checks. A process with elevated permissions can monitor this system call for keys, extract them and forward them to the NMS.

4.1.3.2 Client-side Filtering of Sensitive Connections

Once the keys have been obtained by the daemon on the endpoint, it has to decide if the keys should be forwarded to the NMS to enable decryption of the associated connection. This filtering process considers connections attributed like IP addresses, hostnames and domains name to mark a connection as sensitive according to both network and user policy. For sensitive connections, the daemon does not transfer the keys to the NMS, forcing it to fall back to metadata-only analysis.

Overall the decision to mark connections as sensitive is governed by:

- **Network policy:** It establishes the baseline for which connections must be decrypted and thus which keys need to be always forwarded. This will oftentimes be a "forward-by-default" policy, as the organization naturally wants as much visibility as possible. However, there are cases where full visibility must not be required, e. g., legal or compliance scenarios in which certain information is not allowed to be decrypted. An example would be the healthcare-sector where medical data is transferred through the network. Such scenarios require a more sensible network policy, that enforces visibility for selected services only and cannot use existing state-of-the-art MitM proxies that decrypt all TLS traffic.
- **User policy:** It compliments the network policy by allowing the user to opt-out of decryption for selected services that are not covered by the network policy already. The user policy can be implemented via a self-serve portal where users can submit opt-out requests for certain domains (or in special cases IP addresses/hostnames). After review the updated policy is installed on the endpoint. Naturally, the user policy has to be sanctioned by security personnel and thus will likely not allow the user to retain privacy for all desired services. In these cases the network policy would likely overrule the user choice. However, the conceptual mechanism to include user choices itself already offers a large advantage compared to MitM proxies that decrypt all communication without any direct user involvement.

The concrete filtering process can be implemented via cryptographically signed policy lists that are validated via a network internal public-key infrastructure (PKI). Each connections' attributes are matched to identifiers in the policy lists and marked accordingly. The daemon then transfers keys for non-sensitive connections to the NMS. Validation failure of the list integrity implicates unauthorized modification and could even be used as an additional indicator of compromise (IoC). In this case, the daemon would fall back to forwarding all keys to the NMS to aid security personnel during investigation of the incident.

Overall, we recognize that selective decryption is (*i*) not feasible in all scenarios (especially high-security environments) and (*ii*) may impose additional challenges on security personnel. However, it first and foremost offers a choice to organizations to support users in retaining some privacy which is not possible with the current state-of-the-art. Moreover, we believe, that the improved user privacy can outweigh the disadvantages in many scenarios or even enable the use of NMS in certain scenarios that have previously not been possible because of legal or compliance reasons, e. g., medical institutions. Finally, especially in times of increasing home-office work, the lines between private and work activity are blurred and often crossed. In such cases, users have legitimate reasons to conceal certain private connections from the organization's NMS which is enabled by our approach.

4.1.3.3 Transferring Keys to the NMS

After the filtering process is completed or skipped, the key material is forwarded to the NMS. The actual transmission method used is dependent on the NMS in question and conceptually trivial. However, the transport channel should be chosen carefully, as the key material is highly sensitive. Mutually authenticated TLS connections also based on the internal PKI between endpoints and NMS are a solid choice. The transmission should be direct without additional intermediaries to minimize the threat surface and

prevent unauthorized decryption outside of the NMS. However, if the network policy requires it, keys could be also stored to a secured database to perform authorized offline decryption of dumped traffic at a later point in time.

4.1.3.4 Passive TLS Decryption at the NMS

Once the NMS receives TLS key material from the endpoint, it needs to identify the connection the key material belongs to. This identifier differs for TLS versions and different implementations of session resumption. To identify a connection that was established via a full TLS 1.2 handshake, as shown in Figure 2.2, the *client random* that is sent with the *ClientHello* uniquely identifies the connection and thus can be used to match key material. For subsequent resumed connections, the NMS then needs to also associate the key material with the session identifier. This is either *session ID*, *session ticket* or *TLS 1.3 PSK* depending on TLS version and implementation. If the session identifier is then encountered in a resumed handshake, the NMS can match the key material as for full handshakes. In TLS 1.2 this process is trivial as both session ID and session ticket are sent in cleartext and thus can be easily stored. The TLS 1.3 PSK approach is slightly more complicated as the PSK is already encrypted when sent from the server in the first connection and additionally exchanged after the first use in the resumed connection. However, this does not pose a problem to our approach as the NMS can already decrypt the first connection with key material from the client and thus also store the encrypted PSK for later use.

Once the connection has been identified, the NMS uses the obtained secrets to decrypt TLS application data packets in the connection. This process poses two major challenges: (*i*) **Cipher diversity:** The different TLS versions support varying cipher suites that the NMS needs to implement. This can be addressed by reusing popular cryptographic libraries such as OpenSSL [TYH22] or NSS [NSS22]. An incremental approach with a subset of supported cipher suites is also possible starting with popular ciphers. (*ii*) **Missing key material**: Key material might not be available, once the first encrypted packet of a TLS connection arrives. This can happen either because the endpoint withheld the key material due to filtering or because the key material is simply still in transit to the NMS. In the first case, the NMS should simply skip decryption, while the second case would require the NMS to buffer packets until the key material arrives. However, this might introduce a denial of service (DoS) attack vector if the NMS needs to buffer too many packets for connections especially as the NMS cannot know in advance if key material will arrive at some point in the future. To solve this problem, a traffic buffer in front of the NMS should be deployed that delays only the encrypted traffic for a short time before it is forwarded to the NMS. This way keys can arrive in time for complete decryption. Connections for which no key is present at that time are considered filtered and thus not decrypted. We quantify the impact of the key transfer latency on the decryption and recommend a buffer size in our evaluation (see: Section 4.1.4.4).

4.1.4 Discussion and Evaluation

In this section, we first discuss the computational complexity our approach induces and compare it to the commonly deployed TLS intercepting MitM proxy. Next, we describe the features and limitations of the proof-of-concept (PoC) prototype we implemented to test and evaluate our concept. Finally, we describe the results from two experiments in which we measured the *decryption overhead* and the impact of *key transmission latency* on the decryption success rate, respectively.

Approach	CV	KE	E_{sym}	D_{sym}
TLS Client	1	1	s	r
TLS Server	0	1	r	s
Intercepting Proxy	1	2	$s + r$	$s + r$
Decrypting and passive NMS	0	0	–	$s + r$

Table 4.1: Simplified computational complexity for typical TLS connections

4.1.4.1 Computational Complexity

We compare the computational overhead of our approach on a conceptual level with intercepting MitM proxies. TLS adds non-negligible computation overhead to every connection and the proxy has to setup and maintain two TLS connections per endpoint connection. A decrypting NMS remains passive and thus does not directly initiate TLS connections. Instead, it only decrypts the encrypted payloads to gain access to cleartext data.

Table 4.1 summarizes the simplified computational costs of establishing TLS connections for the TLS client and server respectively, but also for a MitM proxy and a passive NMS. We assume the default behavior in HTTPS where the client authenticates the server only. The TLS client performs a *certificate validation* (CV) for the server certificate and a *key exchange* (KE) to establish session keys. The exact number of cryptographic operations here differs based on the cipher suite used, but usually involves CPU intensive asymmetric cryptography. Once the key material is established, the client encrypts the s bytes of its request via the symmetric cipher E_{sym} that was selected in the handshake and sends them to the server. After a response is received, the r number of bytes are decrypted. The TLS server performs a similar number of operations, but usually without certificate validation as TLS clients rarely use certificates in HTTPS. The number of encrypted and decrypted bytes are switched as request bytes from clients s have to be decrypted and response bytes r encrypted. However, as a symmetric cipher is used with identical costs of encryption and decryption, the overall computation effort should be nearly identical.

The MitM proxy acts as TLS server to the client and as the client for the requested TLS server and thus performs the combined computations of both endpoints. While the KE results in a predictable amount of computation, s and r vary in different scenarios. For small connections such as simple websites, the doubled amount of symmetric cryptography is less impactful, while large connections such as a large file download via HTTPS introduce significant amounts of additional computation on the proxy. The decrypting NMS in our approach has one large conceptual advantage to the MitM proxy: due to its passive analysis, it only needs to decrypt all bytes $s + r$ of a single connection once to achieve cleartext access to the payloads. As no TLS connection is actively established, no KE needs to be performed and even validation of the server certificate is not strictly required as the client would rightfully reject the connection in case of failure. However, analysis of the server certificate might be interesting as a regular NMS analysis outside of decryption.

Additionally, our approach offers some further potential performance gains compared to MitM as the NMS can decide on-demand to skip decryption if cleartext data is not required. This might make sense for common use cases like video-conferencing or streaming services (if allowed at all in the network) which are notorious for their high

bandwidth demands and thus comparatively high decryption cost. While a MitM proxy in theory could skip decryption of the associated connections, e. g., by detecting the domain and transparently forwarding the *ClientHello* instead of terminating it, the NMS can simply skip decryption as it is only a passive part of the connection. This can help to reduce D_{sym} below the maximum of $s + r$. Furthermore, our approach offers yet again greater flexibility, as the NMS can adapt this decision based on the current threat level or other intelligence, e. g., by enabling decryption of all available connections if a current attack is taking place or other imminent threats are expected.

In summary, our passive approach conceptually requires less computational intensive operations when compared with the commonly deployed approach of TLS intercepting MitM proxy servers. While symmetric cryptography is continuously becoming faster to execute on modern hardware due to special instruction sets, it still represents the largest share of computation in a proxy and thus dictates how machine resources need to be scaled for high-bandwidth deployments. Our approach requires significantly less computation and greater flexibility about which connections to decrypt at all while still enabling the NMS to perform analysis on cleartext payloads. However, the lowered computational load is shifted from proxy machines to NMS machines which have to be scaled up accordingly. Thus, our first experiment, that we describe in Section 4.1.4.3, aims to quantify this impact of added decryption on NMS performance to estimate the required changes for real-world deployments.

4.1.4.2 Implementation of Prototype

We implement our approach for passive and transparent TLS decryption as a proof-of-concept prototype for the Zeek NMS [Pax99]. While some features were intentionally left out to reduce complexity, our prototype is able to successfully decrypt TLS connections and perform analysis tasks on the cleartext application data payloads in real-world scenarios. Our prototype currently consists of a patched Zeek version based on v3.2.3 and a simple Python daemon that runs on endpoints and forwards key material. Both TLS key derivation and the actual decryption is performed by `openSSL`, as Zeek already links to this library.

Obtaining TLS Key Material on Endpoints For simplicity, we *obtain key material* via the SSL keylog interface as described in Section 4.1.3.1. We instrument the Firefox browser to write TLS pre-master secrets and client randoms to a file monitored by our Python daemon which are then forwarded to Zeek. Additionally, we configure Firefox to offer `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384` as the only supported cipher suite in the TLS handshakes to ensure that our Zeek version can decrypt the established connections. While we opted for Firefox, our setup it not limited to this browser, as other popular browsers like Google Chrome can be configured to dump TLS pre-master secrets in the same format.

Client-side Filtering of Sensitive Connections The daemon currently does not perform any filtering and thus does not support *selective decryption*. Instead all client randoms and pre-master secrets are forwarded to Zeek independent of the domains involved. A complete implementation would need to match connections identifiers and secrets to domains as this information is not directly present in the SSL keylog file alone. This could be implemented in a browser extension (as the two data points can be linked there) or via direct modification of the cryptographic library in use. This

would enable all applications that use the patched library to be compatible with our approach.

Transferring Keys to the NMS The Python daemon maintains a connection to the Zeek instance via *broker*, Zeek's default communication library, and *transfers new secrets to Zeek* once they appear in the file.

Passive TLS Decryption at the NMS Our patched Zeek version is able to receive both pre-master secrets or derived TLS session keys. In the first case, the TLS key derivation is performed once the first packet of the matching connection is encountered and the resulting session keys are cached for later use. Once session keys are available, Zeek *decrypts the TLS application data* and forwards the cleartext to its internal protocol detection engine and subsequent analyzers.

Current Limitations Our prototype was explicitly built as a proof-of-concept. It is capable of passively decrypting real-world TLS traffic without actively intercepting the connection. However, it currently exhibits some limitations that prevent complete real-world deployments: (*i*) Our prototype only supports a single, but actually one of the most popular, cipher suite, namely `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`. This cipher suite was chosen as it is commonly used in TLS 1.2 and exhibits several features of a modern cipher suite like perfect forward secrecy. It uses the ECDHE key exchange with RSA for authentication and AES in GCM for symmetric encryption of payload data. GCM is favorable to our prototype as it was designed with parallel processing in mind and allows each block to be decrypted separately. Thus, our prototype is in worst-case also able to start the decryption in the middle of the connection if some initial packets could not be decrypted. (*ii*) Our prototype does not perform any *traffic buffering* either directly in Zeek or elsewhere and thus may encounter an encrypted packet before the respective keys are received. If this happens, it prints a debug message and skips decryption for this packet only. This results in partial decryption of the overall connection. Usually, this means that the contained HTTP/2 data cannot be fully parsed and analyzed. This limitation is however not directly tied to our implementation, as a real-world deployment likely would not buffer packets in the NMS itself anyway (as this opens up a potential DoS vector) but rather by a dedicated hardware or software appliance in front of it. The required size of this network buffer to avoid missing the beginning of the connection is an important question. Thus, we estimate this size for small networks in second experiment, that we describe in Section 4.1.4.4. (*iii*) Support for *TLS session resumption* is currently not implemented, but can be added by linking the respective connection identifier (Session IDs, tickets or PSK) to the received key material. The different identifiers are encountered in varying parts of the connections and may even be encrypted. However, the NMS retains complete visibility into all payloads (if key material is available in time for the first packet) and can thus retrieve and store the relevant identifiers. While it was left out in our prototype to reduce complexity, it should definitely be supported in a full implementation, as TLS session resumption is used widely across the web especially for large content delivery networks (CDNs).

Summary In summary, we build a prototype that serves as a proof-of-concept NMS and that is able to decrypt TLS traffic in a passive manner. Given either pre-master secrets or the respective session keys, our patched Zeek version is able to analyze TLS-encrypted HTTP/2 traffic as if it were captured in cleartext. Currently, our prototype

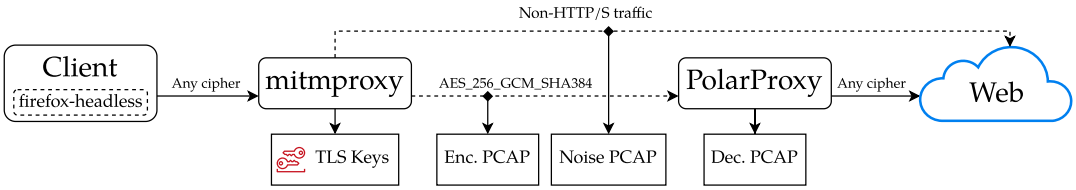


FIGURE 4.2: Experimental Setup: Dataset capture for decryption overhead

exhibits some limitations that prevents it from real-world usage. However, these limitations, such as lack of cipher suite diversity or missing support for TLS session resumption, are mostly engineering-focused problems that can and will be solved with time and existing concepts and libraries. Furthermore, the addition of new cipher suites should not negatively impact performance while adding (and using) selective decryption through filtering increases performance as the NMS needs to decrypt less bytes. Thus, our results obtained in our experiments should provide a solid indication of the performance of a full implementation. The prototype is fully available on GitHub[1] and meanwhile has been merged into Zeek mainline [AW22].

4.1.4.3 Experiment I: Decryption Overhead

In this first experiment we evaluate the central performance metric for our prototype implementation: *decryption overhead*, i. e., the additional runtime added by performing TLS decryption. This metric is important when deploying TLS decryption, as NMS machines need to be scaled up to accommodate for the additional computation requirements. There are two interesting scenarios to consider:

1. In the first scenario, an organization is currently not deploying MitM proxies to terminate and decrypt TLS connections. Instead, a regular NMS is used to analyze encrypted traffic at the network boundary. Due to the encryption, the NMS is only able to analyze connection meta-data resulting in a comparatively low resource utilization.

2. In the second scenario, the organization follows industry best-practices and deploys a MitM proxy that forward payloads from terminated TLS connection to a NMS. In this case, the NMS has full cleartext access to all connections and can perform a more sophisticated analysis, which results in higher resource utilization.

In both scenarios the NMS machines need to be scaled up, as more computation will be performed. However, in the second scenario, MitM proxies are no longer required and their previously allocated resources can be used for monitoring. To evaluate the decryption overhead for both scenarios, we design a capture environment that simulates both deployments.

Setup and dataset Figure 4.2 shows the setup we used to capture the dataset for this experiment. The client machine runs Firefox in headless mode that visits all Alexa Top 1000 websites one after the other. The traffic is transparently routed through `mitmproxy` [CHK21] to extract the TLS pre-master secrets and force the cipher suite that our prototype implements. The outgoing traffic from this machine is then routed differently based on the destination port. Web traffic destined for the ports 80/443 is

[1] see: https://github.com/UHH-ISS/zeek

forwarded to another host running `PolarProxy` [NET21], while all remaining non-web traffic is routed directly to the Internet. We use PolarProxy as it can export decrypted HTTP/2 traffic directly to a pcap file, a feature that `mitmproxy` does not support. This pcap represents the commonly deployed second scenario described above in which the NMS receives decrypted traffic from a TLS intercepting proxy. The encrypted traffic and noise traffic is also captured and stored in different pcap files. These pcaps represent the aforementioned first scenario in which the NMS analyzes encrypted traffic only.

The resulting dataset captured from our deployment comprises the *TLS pre-master secrets* extracted from mitmproxy and pcaps for *encrypted web traffic*, *other noise traffic* as well as the *decrypted HTTP/2 payloads*. The dataset consisting of pcaps and the captured TLS pre-master secrets are publicly available [WHF22].

Results By using our captured datasets, we can now compare the runtime of both, the unmodified Zeek version and our prototype. Please note that this first experiment focuses on decryption performance and overhead alone. As such, we configure our Zeek prototype to load all TLS pre-master secrets from disk at the start of the experiment and let Zeek analyze the respective scenario pcaps. This results in all keys being instantly available, which is not comparable to real-world deployments. To address this, we analyze the impact of real-time transmission of TLS keys and their delayed arrival at the NMS in the second experiment (see: Section 4.1.4.4).

Figure 4.3 shows the decryption overhead by comparing the runtime of the unmodified Zeek version and our prototype for three different traffic profiles. Each traffic profile is defined in both *decrypted* and *encrypted* fashion as follows: (*i*) **https** contains only traffic that was originally sent via HTTPS. This includes most websites from Alexa top 1000 minus a few that did not support TLS at all. The decrypted variant is built from the decrypted pcap written by PolarProxy while the encrypted variants is derived from the pcap captures between mitmproxy and PolarProxy. All captures are filtered by destination port 443 to limit to HTTPS traffic. (*ii*) **web** consists of the same pcaps but removes the port filter and consists of all HTTP and HTTPS traffic. (*iii*) **all** extends the previous traffic profile by adding non-web noise traffic. The decrypted variant simply merges the decrypted pcap by PolarProxy with the noise pcap and the encrypted variant merges the pcap captured between the proxy machines with the noise pcap.

The first two bars per traffic type show the runtime for the cleartext traffic for both the unmodified Zeek and our prototype. For all three traffic types, the runtime is nearly identical. This is expected as no decryption has to be performed and both version perform identical analysis tasks on cleartext data. These measurements thus resemble the NMS runtime in a typical MitM proxy deployment without including the runtime of the proxy itself. The third bar shows the runtime of the unmodified Zeek version analyzing encrypted traffic. The runtime is smaller than for the cleartext traffic, as this Zeek version cannot decrypt the payloads and thus falls back to metadata only analysis. This difference is most visible for the HTTPS dataset as no noise or HTTP data is present, which would be in cleartext and thus potentially triggering complex analysis scripts. This measurement also establishes a baseline for the first scenario described in the introduction of this section: a NMS processes encrypted network traffic and no MitM proxy is deployed. The fourth bar shows the runtime our prototype that decrypts the TLS connections and analyzes the contained payloads. The added decryption results in a comparatively large runtime overhead as expected. Compared to the MitM baseline (cleartext traffic), we see an increase by a factor of 2.5.

FIGURE 4.3: Results: Decryption overhead for passive TLS decryption

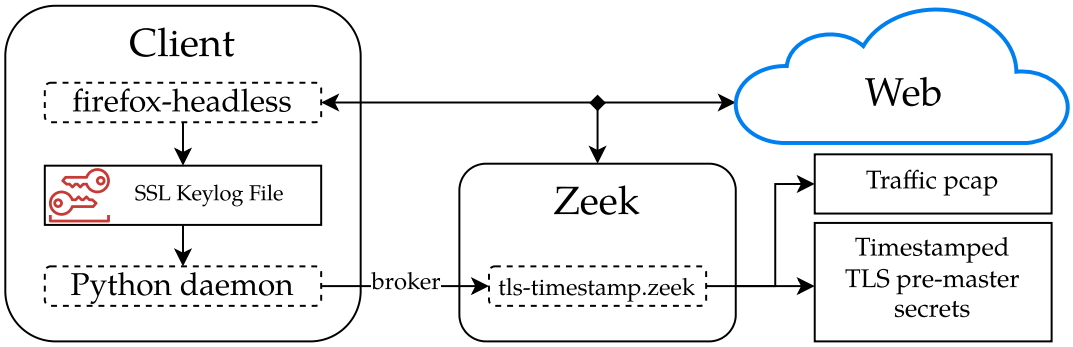


FIGURE 4.4: Experimental Setup: Dataset capture for key transmission latency

An administrator should scale the resources of NMS machines by this factor when switching from an MitM proxy deployment to our approach. As MitM proxies are no longer needed in this scenario their resources could be repurposed. Compared to the first scenario (encrypted traffic), the overhead grows to about 3 times. However, this is reasonable, as no decryption was performed previously (neither by a MitM proxy nor the NMS). The previous comparison and our discussion in Section 4.1.4.1 show that deploying a MitM proxy would likely result in higher overall resource usage.

Note: While this experiment can be considered a micro-benchmark, especially as our prototype currently only supports a single cipher suite, the results should give a good estimation of the expected decryption overhead in most scenarios. Especially as the missing features are not expected to negatively impact performance. Combined with the previous discussion, we are confident that our approach reduces overall computation efforts compared to a MitM proxy deployment.

4.1.4.4 Experiment II: Key Transmission Latency

In the second experiment, we evaluate the real-world viability of our approach by examining the delay between the key material being made available on the client machine and the arrival at the NMS after transmission through the network. This is

especially relevant, when considering cipher suites that require the cleartext of the previous packet when decrypting a newly arriving packet. For these ciphers either the key material needs to be available once the first encrypted packet arrives, the NMS needs to buffer packets, or the network traffic needs to be delayed before being passed to the NMS.

Setup and dataset We design a deployment to capture timestamped TLS pre-master secrets for this experiment that we show in Figure 4.4. The client machine again runs Firefox in headless mode and is configured to only offer the prototype's supported cipher suite. We then browse the Alexa top 1,000 websites and write the SSL keylog file as usual. Our python daemon also runs on the client machine and watches the keylog file for changes via `inotify`. New pre-master secrets are immediately forwarded to Zeek via its `broker` communication library. On arrival, Zeek timestamps the key material to document the arrival time. Additionally, the machine running Zeek is used as the gateway for the client host which allows Zeek to capture the traffic as a pcap. The resulting dataset comprises the *traffic pcap* as well as the *timestamped TLS pre-master secrets*. It is also available as part of [WHF22].

Results For our second experiment, the timestamped TLS pre-master secrets are again pre-loaded in our prototype that then analyzes the captured pcap file. However, the timestamp now dictates when Zeek starts to actually use the key. For the default case, it only starts decryption once the network timestamp in the pcap is larger than the timestamp of the corresponding pre-master secret. This resembles a real-world deployment without any traffic buffering. For simulating this traffic buffering, we have two options: (*i*) we can alter the timestamps in the pcap file by the respective time interval while leaving the pre-master secret timestamps untouched. This closely resembles delayed traffic. (*ii*) we can use the pcap as is and simply add the desired time interval to the current network time in the comparison operation in Zeek. For simplicity we chose the second option as the delay can easily be configured in a script. In the experiment runs, we count the number of TLS payload bytes that could be decrypted. As our implemented cipher suite can start to decrypt from any packet in the connection, this can be anything between 0 (if the key arrives after the connection is already finished) or the full amount of payload bytes in the connection. Additionally, we consider a connection completely decryptable if the key is available before the first encrypted payload packet is encountered. This is important for two reasons: (*i*) some cipher suites can only decrypt all or no packets in a connection as packet payload is used as IV in decryption of the next packet. (*ii*) especially for HTTP/2 the first bytes of the connection are important for the NMS as they contain the client request. If they are missed, the NMS cannot analyze the protocol level. The *decryption performance* is then calculated as the ratio of decrypted bytes/connections and total bytes/connections respectively.

Our results are shown in Figure 4.5. The x-axis shows the simulated traffic delay in milliseconds and the y-axis the decryption success rate, i. e., the respective percentage of bytes and connections that could be correctly decrypted. In a real-time deployment and without any delay only 1.3% of all connections are completely decryptable. This is expected as the key transfer latency prevents the first packets from being decrypted. However, 98.9% of all TLS payload bytes can already be decrypted. This indicates that already only few of the first bytes are missed. This is confirmed in the first few 1ms steps as the success rate for connections increases sharply. At 5ms traffic delay already 65.3% of all connections can be decrypted while 10ms delay results in a success rate of

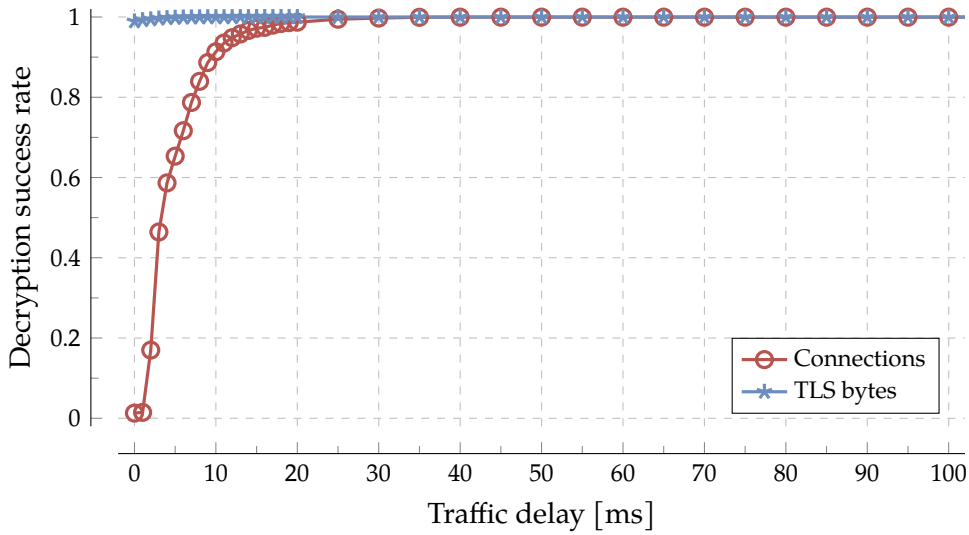


FIGURE 4.5: Results: Decryption success rate depending on traffic delay

91.3% for connections and 99.96% of TLS bytes respectively. The success rates steadily increase until at 40ms effectively 100% of both bytes and connections can become decrypted.

The results from our second experiment indicate that only a small delay is required (in our setting 40ms) to achieve nearly complete coverage for decryption. For real-world deployments, traffic can be sufficiently buffered before passing it to the NMS. Depending on the utilization of the tapped network link, this results in a buffer size of up to 400M for a typical 10G uplink in a medium sized network. It is important to note that this small delay is only required before passing traffic to the NMS and thus does not impact other hosts in the network. Problems may arise if the NMS is also used as an intrusion prevention system (IPS) and thus controls a firewall. In these scenarios, the delay may result in the IPS reacting too slow to effectively block intrusion attempts. Especially in cases like these, it might be advisable to implement the buffering as part of the TLS parsing layer of the NMS itself—which is more complex, but in turn does not increase reaction latency.

In summary, the discussion and evaluation of our approach for passive TLS decryption for NMSs show promising results especially compared with the current state-of-the-art in the form of TLS-intercepting MitM proxy servers. Our approach is conceptually less demanding in terms of computations as communication data only needs to be decrypted once. Our experiments demonstrated that overhead imposed by decryption is moderate with about 2.5 times total runtime while the latency of key material transmission can be addressed by introducing a small traffic buffer as small as 40ms in front of the NMS.

4.1.4.5 Requirement Comparison

Based on the description and evaluation results, we can now compare our approach for passive TLS decryption via cooperative endpoints with our requirements formulated in Section 3.1. As this approach does not perform any kind of intrusion or APT detection, *R1: Accuracy* and *R2: Explainability* are not applicable and thus not rated. Our approach fulfills *R3: Low overhead* partially as there are less required computations compared to MitM proxies, although endhosts need to run an agent to forward the key

material. *R4: Scalability* is met since our approach can scale horizontally with more available resources for decryption. In addition, it is easier to distribute load across multiple hosts running an NMS compared to the conceptually single proxy host. Due to the lesser attack vector exposed by a passive NMS compared to an active MitM proxy *R5: Security* is fully checked. This is especially true for non-AEAD ciphers as our approach can limit the potential damage inflicted by a compromised NMS even further in this case. *R6: Privacy* is fulfilled as our approach can leverage selective decryption to conceptually let users retain some privacy for selected websites. While some scenarios do not allow for this, our solution is likely optimal under the circumstances. Lastly, *R7: Deploy- & Maintainability* is partially checked as our approach relies on an important infrastructure of cooperating endhosts that need to run our daemon to obtain reliable visibility.

4.1.5 Summary

This section presented an approach for passive TLS decryption on trusted NMSs via key sharing from cooperative endhosts to answer our formulated research question **RQ1**:

RQ1 *How can NMSs regain visibility into TLS-encrypted communication without actively intercepting the connections on a central entity and breaking end-to-end guarantees?*

This visibility is essential as workstations are often the initial attack vector of APT campaigns. Our approach shows how cooperative endhosts can alleviate the need for a central intercepting entity as the key material is proactively transmitted to the NMS such that it can decrypt the communication passively. Compared to TLS interception via MitM proxy servers, we discussed how our approach (*i*) requires less computations, (*ii*) keeps the security guarantees of the original TLS connection intact and (*iii*) conceptionally reduces the threat surface as absolute trust in a single entity (i. e., the MitM proxy and its certificate authority) is no longer required. We implemented our approach as a Zeek [Pax99; Zee22] module that is able to receive keys from endhosts and decrypt the corresponding TLS connections. In our evaluation, we measured the additional overhead induced by adding decryption to the NMS as well as the impact of key transfer latency on the decryption success. Our results indicate that a moderate increase of computational resources of about 2.5 times is required while a small network buffer of 40ms is sufficient to bridge the time required for key transfer in our testbed. Additionally, our Zeek prototype has been extended and merged into Zeek mainline as of version v5.0.0 [AW22]. Overall, our approach described how passive TLS decryption can be achieved without actively intercepting the connection via a MitM proxy and thus provides an answer to **RQ1**.

This contribution enhances security monitoring via improved internal visibility into TLS-encrypted payloads. The next section introduces a mechanism for additional external visibility through characterization of brute-force attacker behavior. The approach leverages a collection of both established and novel metrics to cluster and prioritize brute-force attackers to support SOC analysts in their investigation.

4.2 Characterization of Brute-Force Attackers

This section introduces an approach to characterize behavior of brute-force attackers that target services that are available from the open Internet. This is important as large parts of these malicious login attempts are caused by simple automated scans

that are (comparatively) harmless. However, APT adversaries may attempt a limited number of logins as part of their reconnaissance efforts. Thus, SOC analysts need to carefully investigate these attempts. Overall, this contribution aims to answer our research question **RQ2** as introduced in Section 1.1:

RQ2 *How and to what extend can brute-force attempts on externally-accessible services be categorized and screened for potential APT reconnaissance activity?*

To address this problem, we discuss metrics established in previous literature and introduce two novel ones that can be leveraged to characterize brute-force attacker behavior. This is useful for both estimating the attacker's sophistication level as well as to detect potential collusion between otherwise unrelated IP addresses. The resulting characterization helps analysts to distinguish between low-effort/high-volume attempts (that are less likely to succeed) and targeted attempts (that could signal the start of a larger APT campaign). The content of this section is based on the corresponding publication [WF20b] and shares material with it.

4.2.1 Motivation and Objectives

Every service on the open Internet gets probed by malicious login attempts that range from automated scripts continuously trying default passwords on every available IP address to targeted login attempts that may use stolen credentials to access internal services. For organizations that poses a problem, as certain services need to be exposed to the Internet (to be available for customers or remote workers) but doing so opens up a potential attack vector. Due to the flood of automated attempts it is tempting to ignore any alerts generated from brute-force login attempts to lower the load on analysts that need to sift through them. However, this strategy also ignores the targeted attempts in between that in the worst case might be the beginning of an APT campaign. As no ground truth about attacker behavior can be obtained and every attack is different, the analyst has to adapt to a changing threat landscape and needs to infer as much as possible from the data available. While there are approachhes that aim to simply detect brute-force attacks (as we discussed in Section 3.6.1), they focus on distinguishing between failed benign login attempts and malicious attacks and are thus not applicable to APT campaigns. Additionally, standard metrics for login attempts (as proposed in [OM08; ABvO16]) are helpful but fail to capture certain complex attack behavior. To the best of our knowledge, there are no structured threat hunting approaches for brute-force attack scenarios that aim to systematically estimate attacker behavior and sophistication. To address this open challenge, this contribution aims to achieve the following *objectives*:

1. Survey existing metrics for brute-force login attempts from related work, that give insights about attacker behavior.
2. Introduce novel metrics with the same goal that maximize information gain from the limited data present in the login attempts.
3. Discuss and evaluate how the combined set of metrics can be used in isolation or in selected pairs to support analysts, i. e., as indicators for estimated threat-level or potential collusion between multiple IP addresses, which they can leverage to filter, prioritize, and mitigate attacks.
4. Evaluate these threat-level indicators based on real-world data that is obtained from a host connected to the open Internet.

5. Publish the resulting dataset to foster further research in this area.

The remainder of the section is structured as follows: Section 4.2.2 briefly explains the underlying formal model that is used throughout the section. Section 4.2.3 describes established metrics to characterize brute-force attacks that have already been proposed in the literature. Section 4.2.4 proposes novel metrics that aim to capture brute-force attacker behavior in greater detail based on a notion of brute-force sessions and dictionary entropy. Next, Section 4.2.5 highlights how the combined set of established and novel metrics can be used to characterize attacker behavior. In Section 4.2.6 we evaluate our approach on a real-world dataset obtained via our own Honeygrove [Hon+22] honeypot and compare it with the requirements formulated in Section 3.1.

4.2.2 Formal Model

As mentioned in the introduction, the basis for this work are login attempts on publicly exposed services. We formalize this problem as follows: Given the sequence of malicious login attempts L, we aim to identify behavioral patterns of the attacking IP addresses. The definition of L is given in Equation (4.1) using a sequence-builder notation that is similar to common set-builder notation. Each login attempt $l \in L$ can be described by the following features: a timestamp (ts), the source IP address, the protocol the attacker used (pr) and the credentials ($user$, pw). For simplicity, we assume L to be ordered by timestamp ($ts(l_i) \leq ts(l_{i+1})$). Based on this and some metrics, attackers with similar behavior are clustered. Finally, the threat level of both, the clusters or selected attackers inside a cluster, can be estimated by comparing metrics to prioritize defensive countermeasures.

$$L = \left(l_i \mid l_i = (ts, ip, pr, user, pw) \wedge ts(l_i) \leq ts(l_{i+1})\right) \tag{4.1}$$

Although the source IP address can be easily switched by the attacker, e. g., by using proxy servers such as Tor or moving between hosting providers, it represents the basis for all other metrics in this section. As an attacker is much more likely to switch IP addresses rather than change their behavior, the goal is to cluster different IP addresses that belong to the same attacker. For later analysis we group login attempts $l_i \in L$ by source IP. Equation (4.2) formalizes this by introducing the sequence of login attempts per IP L^x that contains all attempts by IP address x.

$$L^x = \left(l_i \mid l_i \in L \wedge ip(l_i) = x\right) \tag{4.2}$$

Note: we assume clear-text passwords. However, most metrics can also be calculated with hashed passwords or to some extent without any password information at all. This enables characterization in real-world deployments where password logging might be turned off for security reasons. Additionally, we assume the data to contain only malicious login attempts as other approaches like [HS19] can be used to distinguish between malicious and legitimate attempts. In these cases it might even be possible to still log clear-text passwords, while discarding this sensitive information for legitimate logins.

4.2.3 Established Features and Metrics

In this section we present the set of features and metrics from previous works, that we believe to be suitable for the characterization of brute-force attackers. We call information that is intrinsic to the login attempts (or that can be derived in very simple ways) *features*. Examples for this would be source IP or the set of credentials used by the attacker. Based on these features we are calculating *metrics*. However, the difference is not always straightforward and as such some features that we show might have been called metrics in previous work. Both, features and metrics, can be divided in three categories, that will be discussed in the next three section respectively:

- *Connection-based* features are based on the netflow attributes of the underlying connection such as IP addresses and protocols used.
- *Credential-based* features and metrics are based on the sequence of credentials used by the attacker and their relation among each other.
- Lastly, *Timing-based* features and metrics are based on the timestamps of the login attempts and derive additional insights off intervals and frequency.

4.2.3.1 Connection-based Metrics

Metadata of the connection the attacker used to perform the login attempts already reveals some information about their origin and characteristics. Furthermore, a set of additional features can be obtained by querying openly available sources. The *Set of Protocols* P^x as defined in Equation (4.3) already offers some insights about their targeted services. In many cases this is SSH as it is the most popular remote shell protocol for Unix-like operating systems. However, an attacker might target multiple protocols at the same time. As shown in Equation (4.3) P^x is obtained by just taking the unique set of protocols across all attempts of the IP.

$$P^x = \left\{ pr(l_i) \mid l_i \in L \right\} \tag{4.3}$$

The *rDNS* entry of attacking host also offers weak indicators and can be easily obtained. While a sophisticated attacker might use a shared, static one (like `no-reverse-dns-configured.com`) or even provide explicitly fake entries, the default configurations of hosters often include a unique name that hints at the company or even the respective customer. The */24 subnet* of an attacker can help to reveal naive attackers that, e.g., use consecutive IP addresses in the same subnet. Furthermore, autonomous system (AS) membership results in AS *number*, *owner* and *country* as additional features to correlate attackers across subnet borders. For attackers that use the same cloud provider for all login attempts either the AS number matches directly or the AS owner is the same. For attackers that use multiple providers the country might still match. Finally, the source IP can be matched against a list of known *Tor* exit nodes. However, this makes correlation extremely difficult as multiple attackers might use the same Tor exit node or a single attacker might switch exit nodes during the session. Both cases might invalidate an analysis that is primarily based on IP addresses. Thus, requests from Tor exit nodes (or other proxy/VPN services if the IP addresses can be obtained) should be removed from the dataset and analyzed manually.

4.2.3.2 Credential-based Metrics

The credentials, that a brute-force attacker uses, are one of the strongest indicators for their sophistication. Two features are relevant when considering these credentials:

- The ordered *Credential Sequence* C^x that contains credential pairs from all login attempts of the attacker which might include duplicates. The definition for C^x is given in Equation (4.4).
- The set of unique credentials pairs (also called *Dictionary*) D^x that is obtained by removing duplicates from C^x as shown in Equation (4.5).

As C^x is directly derived from L^x it is implicitly ordered by timestamp to support meaningful analysis while D^x is by definition an unordered set.

Metrics based on the attacker's credentials are covered in related work, e. g., in [OM08; ABvO16]. Therefore most credential-based metrics are taken from these works. They rely on some form of password information being present in the requests (clear-text or hashed) and are of limited applicability if only usernames are considered.

$$C^x = \Big(\big(user(l_i),\ pw(l_i) \big) \mid l_i \in L^x \Big) \tag{4.4}$$

$$D^x = \big\{ c_i \mid c_i \in C^x \big\} \tag{4.5}$$

The *size* of the attacker's dictionary is one of the simplest metrics and can help to differentiate between naive attackers that use a small set of credentials (such as default credentials of specific IoT devices) and generic attackers that (usually) try a large number of credentials before moving on to the next target. We can formalize this as the cardinality of the dictionary $|D_x|$. Furthermore, it is important how "unique" their dictionary is when compared with the dictionaries of all other attackers. The *Total Dictionary Overlap* $o_t(D^x)$ measures this by calculating the set intersection between the dictionary of the attacker D^x and all other attackers in the dataset. Equation (4.6) calculates $o_t(D^x)$ as the mean of these values across the dataset.

$$o_t(D^x) = \frac{\sum \left(|D^x \bigcap D^i| \mid i \in ip(L) \wedge i \neq x \right)}{|ip(L)| - 1} \tag{4.6}$$

As only dictionaries are compared, the overlap can also be computed for any set of dictionaries. A special case of this is the *Subnet Dictionary Overlap* $o_s(D^x)$. Instead of all other dictionaries this metric is obtained by calculating the mean overlap between all other IP addresses in the same /24 subnet. Related work and our own results show, that some attackers use multiple IP addresses from the same subnet [ABvO16] and this overlap can help to reveal coordinated dictionary splits ($o_s(D^x) = 0$). Additionally, we can consider a known credential leak as a potential dictionary to compare to. We call this metric *Leak Dictionary Overlap* $o_l(D^x)$. A high value for this metric directly implies a lesser threat level for this attacker, as they did not attempt custom tailored credentials but instead relies on leaked credentials that should have been blocked in the organization anyway.

Finally, we use two boolean metrics to further analyze the complexity of the dictionary. We check, if an attacker uses duplicate credentials in either the same *session* (see next

section) or at all. The first hints at either a bug in the attacker toolkit or a poorly designed credential list, while the second might be the result of an attacker returning to the same host and retrying credentials.

4.2.3.3 Timing-based Metrics

The timing of login attempts might also provide some insights into how an attacker potentially distributes their brute-force attempts. With multiple points of data collection as presented in [ABvO16], it is possible to also consider the time it takes the attacker to move between the targets and thus estimate the size of the scanned IP range. We chose the *Login Frequency* $\nu(L, x)$ (as given in Equation (4.7)) for our approach as it captures the amount of investment the attacker devotes towards the target. However, this metric does not account for persistent attackers, that return to the target multiple times. Thus, it scores similar values for an attacker that performs a large amount of attempts in one session and for an attacker that carries out a smaller number of attempts each day.

$$\nu(L, x) = \frac{|L^x|}{ts_{max}(L) - ts_{min}(L)} \tag{4.7}$$

4.2.4 Novel Metrics: Brute-Force Sessions and Dictionary Entropy

We introduce two new metrics to provide additional insights about the attacker's behavior, namely *Brute-Force Sessions* and *Dictionary Entropy*. In this section, we describe both concepts, derive useful metrics and highlight the impact on both classification and prioritization of attackers in a defensive context.

4.2.4.1 Brute-force Sessions

The timing of a sequence of login attempts offers insights about the attacker's strategy. To capture these sequences and analyze them in conjunction, we introduce the concept of *brute-force sessions*—the set of login attempts that were observed within a certain timeout τ after each other. This is visualized in Figure 4.6: in the first line, τ is set to a shorter value and the first session S1 ends accordingly after l_1 as the timeout is exceeded between l_3 and l_4. However, if τ is set to a higher value, as shown in the second line, S1 is extended to include l_3. This grouping and related metrics enable us to differentiate between, e. g., an attacker that scans each IP address in their search space once a day with exactly fifteen credential combinations and an attacker that scans each target only once every two weeks with a massive credential list of 20 000 different combinations.

Given an IP address x, an ordered sequence of malicious login attempts by that IP address L^x and a *session timeout* τ, we can compute a set of scanning sessions as shown in Equation (4.8). τ hereby represents the duration threshold after which the following login attempt is considered to start a new session.

$$S^{x\tau} = \left(\left(l_i \mid l_i \in L^x \wedge ts(l_i) - ts(l_{i-1}) \leq \tau\right)\right) \tag{4.8}$$

The choice of τ impacts the size of S^x. Higher values of τ results in fewer overall sessions as smaller sessions are merged. Depending on the attacker's strategy and τ, all their login attempts might be grouped into a single session. However, the choice

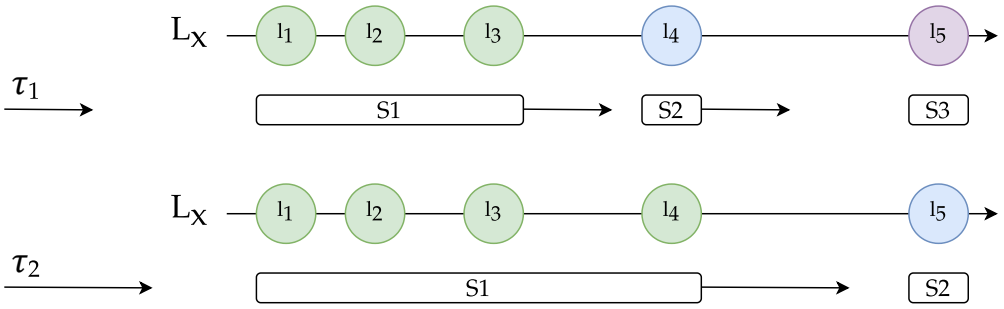


FIGURE 4.6: Example: Brute-force sessions. The parameter τ defines when a new session is started. In the first line, this happens after l_3, in the second line l_3 is included in S1 and the session ends after l_4.

of τ does not impact runtime performance nor memory demands. Thus, small values of τ should be preferred to avoid incorrect merging. In Section 4.2.6.1 we give a recommendation for τ based on the results from our real-world dataset.

Based on $S^{x\tau}$ we define several related metrics to characterize attackers based on their session behavior. The *Session Frequency* $\nu_s(L, x)$ represents the mean number of sessions an attacker initiated in a given timespan. Depending on the duration the dataset was obtained in, this can be seconds, days or even months. $\nu_s(L, x)$ can be calculated by dividing the session count $|S^{x\tau}|$ by this duration. Furthermore, we measure the number of login attempts in a session as *Session Length* $l(S^{x\tau})$ (see Equation (4.9)) and the time in seconds between first and last login attempt of a session as *Session Duration* $d(S^{x\tau})$ (see Equation (4.10)). Based on these two integer sequences, we compute mean (represented by $\overline{x}$) and standard deviation (s) to estimate how (ir-)regular the attacker behaves when considering their timing.

$$l(S^{x\tau}) = \left(|s_i| \mid s_i \in S^{x\tau}\right) \tag{4.9}$$

$$d(S^{x\tau}) = \left(ts_{max}(s_i) - ts_{min}(s_i) \mid s_i \in S^{x\tau}\right) \tag{4.10}$$

Two additional metrics aim to analyze more subtle timing behavior in a session as well as amongst sessions. The *Time-between-Logins* (*TbL*) ($\delta_l(S^{x\tau})$) describes the time that passes between successive login attempts in a single session and is formalized in Equation (4.11). In combination with the session length this allows to differentiate between attackers that quickly scan (often with a predefined set of credentials) and ones that try to stretch out their login attempts over larger timespans to evade detection. The metric calculates a sequence of length $|s_i|$ for each brute-force session in $s_i \in S^{x\tau}$.

$$\delta_l(S^{x\tau}) = \left(\left(ts(l_{i+1}) - ts(l_i) \mid l_i \in s_i\right) \mid s_i \in S^{x\tau}\right) \tag{4.11}$$

The *Time-between-Sessions* (*TbS*) (as shown in Equation (4.12) is a similar metric that describes the time that passes between consecutive sessions of the same attacker. Hence, this metric describes how quickly an attacker returns to the same target.

$$\delta_s(S^{x\tau}) = \left(ts(s_{i+1}) - ts(s_i) \mid s_i \in S^{x\tau}\right) \tag{4.12}$$

4.2.4.2 Dictionary Entropy

We also introduce a new metric to measure the complexity of the credential sequence C^x and how it evolves over time. The *Dictionary Entropy* E^x is the sequence of cumulative Shannon entropies H [Sha48] of C^x. Equation (4.13) formalizes this concept using sequence indexing notation. Given a credential sequence $C^x = (c_0, c_1 \dots, c_n)$ the dictionary entropy is defined as $E^x = \Big(H(c_0, c_1), H(c_0, c_1, c_2), \dots, H(c_0, \dots, c_n)\Big)$. This means that the entropy is increasing as long as new credentials are attempted, that use different characters. For highly similar credentials or duplicates E^x will decrease.

$$E^x = \left(H\Big((C^x_k)^i_{k=1}\Big) \mid i \in \mathbf{N}^0 \wedge i < |C^x|\right) \tag{4.13}$$

E^x is mostly interesting to estimate the complexity of an attacker dictionary. The entropy increases if credentials with different characters are attempted. For credentials that are similar to previously used ones, the increase is lower or might even be negative if only a few characters differ such as number permutations appended to wordlists. Given the two credential sequences C^1 and C^2:

$$C^1 = (\texttt{'root:root'}, \texttt{'admin:admin'}, \texttt{'foo:bar'})$$
$$C^2 = (\texttt{'root:root'}, \texttt{'root:root1'}, \texttt{'root:root2'})$$

the entropy for the first sequence E^1 is higher than for E^2. The `'root:root'` part is present in every credential in the second sequence and thus lowers the increase in entropy after the first credential.

During analysis the *sequence itself* can be used to evaluate the evolution of credential complexity over time, i. e., to detect duplicates and their frequency, while the *mean* acts as a general indicator for prioritization of countermeasures. Additionally, both values can be used for clustering as well.

4.2.5 Characterization and Prioritization

Table 4.2 summarizes all features and metrics discussed in this paper. They can be used to address the two goals in the problem statement, namely:

- grouping of IP addresses/attackers that (partially) share behavioral patterns
- prioritization of countermeasures regarding single attackers or clusters based on sophistication and expected threat-level to the organization

Clustering can be leveraged for both goals depending on the features and metrics used as well as the parameters of the respective clustering algorithm. Running the clustering algorithm with high similarity thresholds, results in clusters of attackers that behave highly similar with respect to the features/metrics selected. However, this potentially results in a large number of clusters and outliers as the clustering is stricter. This is useful for the first goal of grouping attackers/IP addresses. If the threshold is decreased, fewer and larger clusters are formed. Depending on the metrics that are used, these larger clusters are useful for prioritization. The largest clusters usually represent the big part of low-threat attackers with low sophistication and thus can be mostly ignored.

Independent of how clusters were formed, the features and metrics can then be used to calculate intra- and inter-cluster outliers. This is most effective when different

Type	Formula	Name	Description
Feature	L^x	Login attempts	Sequence of all login attempts performed by IP address x
	—	rDNS	Fully qualified hostname (`host.domain.tld`)
	—	Subnet (/24)	Subnet-triple without netmask (0.0.0–255.255.255)
	—	Autonomous System number	Number of the AS the source IP belongs to (1–64495)
	—	Autonomous System country	Country code according to ISO 3611
	—	Autonomous System owner	Company/Organization name (free-text)
	P^x	Protocols	Set of protocols used by the attacker
	—	Is-Tor	`True` if the source IP belongs to a known Tor exit node
Feature	C^x	Credential sequence	Sequence of credentials used by the attacker
	D^x	Dictionary	Set of unique credentials used by the attacker
	$\lvert D^x \rvert$	Dictionary size	Number of unique credentials used by the attacker
	$\lvert C^x \rvert \neq \lvert D^x \rvert$	Duplicate credentials (total)	`True` if C_x contains duplicate credentials
	—	Duplicate credentials (same session)	`True` if duplicate credentials were seen in a single session
Metric	$o_t(D^x)$	Total Dictionary Overlap	Percentage of credentials also seen by other attackers ([0,1])
	$o_s(D^x)$	Subnet Dictionary Overlap	Percentage of credentials also seen in the same subnet ([0,1])
	$o_l(D^x)$	Leak Dictionary Overlap	Percentage of credentials found in previous leaks ([0,1])
	—	Username == password	`True` if an attempt with username == password was seen
	E^x	Dictionary entropy ($\overline{x}/s$)	Shannon-entropies of the words in the credential sequence C^x
Metric	$S^{x\tau}$	Sessions per IP	Sequence of brute-force sessions
	$\nu_l(L,x)$	Login frequency	Mean number of login attempts per day
	$\nu_s(L,x)$	Session frequency	Mean number of sessions per day
	$l(S^{x\tau})$	Session length ($\overline{x}/s$)	Number of login attempts in a session
	$d(S^{x\tau})$	Session duration ($\overline{x}/s$)	Duration between first and last attempt in a session (seconds)
	$\delta_l(S^{x\tau})$	Time-between-Logins (TbL) ($\overline{x}/s$)	Time between subsequent logins in a sessions (seconds)
	$\delta_s(S^{x\tau})$	Time-between-Sessions (TbS) ($\overline{x}/s$)	Time between subsequent sessions (seconds)

TABLE 4.2: Features and Metrics: Characterization of brute-force attackers

Feature/Metric	dataset
Total number of login attempts ($\|L\|$)	1 953 965
Mean number of login attempts per day	~ 65 132
Total number of IPs	14 393
Mean number of login attempts per IP	~ 136
Number of SSH login attempts	1 748 229 (89.47%)
Number of Telnet login attempts	200 125 (10.24%)
Remaining login attempts	5 611 (00.29%)

TABLE 4.3: Dataset Overview: BFL2020

categories are chosen for clustering and prioritization, e. g., credential- and timing-based metrics. For large clusters the analyst might then find intra-cluster outliers based on the cluster mean and for small clusters prioritize them based on inter-cluster comparison of the metric.

4.2.6 Evaluation

In this section we analyze the BFL2020 dataset [WF20a] of brute-force logins that was obtained through Honeygrove [Hon+22]. In comparison other available honeypots [Vas+13], Honeygrove was specifically designed to collect brute-force login attempts for multiple protocols. This implies that all login attempts found in the dataset can be considered malicious. Although we do not have ground truth about the attackers' behavior, sophistication, and threat level the evaluation offers indicators that can be taken into account for manual analysis. It is also important that this analysis neither claims to be complete nor generally applicable for all datasets. The goal is to demonstrate the effectiveness of the selected metrics to obtain (additional) indicators for attacker attribution that can can be used in the context of threat hunting and manual security analysis in a SOC. The exact choice of metrics and algorithm parameters varies from organization to organization and even across time as multiple attackers appear on the threat-landscape.

The dataset consists of about 2 000 000 password-based login attempts on a single Honeygrove instance in a three month period between 2019-12-03 and 2019-06-13. Table 4.3 shows some characteristics of the dataset. It is freely available online to enable further research on the behavior of brute-force attackers.

4.2.6.1 Results of Single-Metric Analysis

The features and metrics shown in Section 4.2.3 and Section 4.2.4 provide a basis for characterization of brute-force attackers. Depending on the attacker's behavior, a distinct combination of metrics might identify them in the pool of attacking IP addresses. In this section, we show results for selected metrics for our dataset and highlight their relevance for distinguishing attacker behavior.

Dictionary size The dictionary size $|D^x|$ is one of the most intuitive features to roughly estimate the sophistication level of the attacker. Figure 4.7 shows the cumulative distribution function (CDF) for this feature in our dataset. Most IP addresses use a relatively small dictionary with about 80% of them trying 30 unique credentials combinations or less. Interestingly the graph rises sharply around a size of 85–86

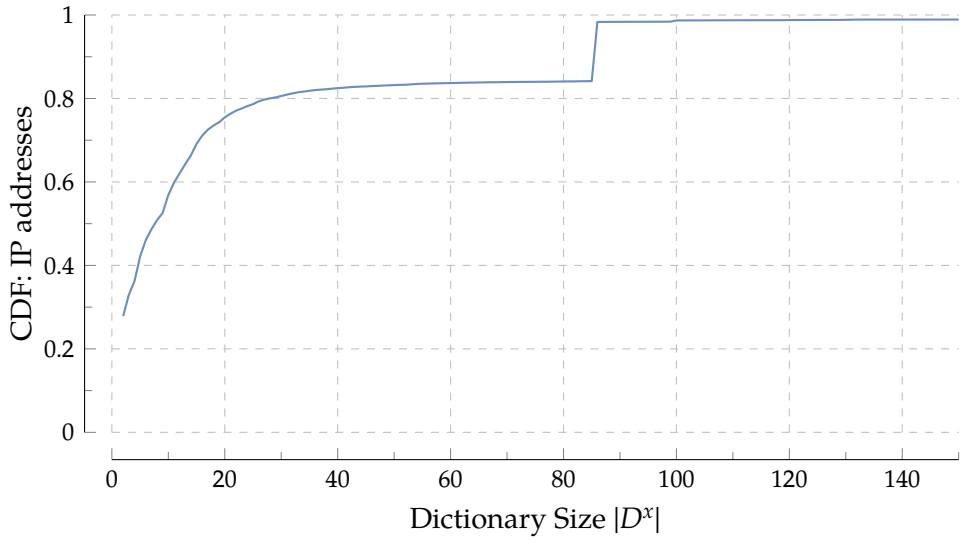


FIGURE 4.7: CDF: Dictionary Size $|D^x|$

indicating that a comparatively large number of IP addresses exhibited that dictionary size. This indicates that either a similar dictionary is used by multiple IP addresses or that this value was often chosen for other reasons such as to avoid detection, e. g., an attacker might switch IP addresses after this number of login attempts. The largest observed dictionary size is 14 120, which is not included in the graph anymore for visibility.

As noted above, the dictionary size alone does allow for an accurate assessment of the attacker's threat level to the organization. A small dictionary might be the result of the guessing of default passwords or a highly targeted attack via credentials obtained from spearphishing. Similarly a large dictionary can be obtained by using a large and complex wordlist or by guessing numeric permutations of a small wordlist such as `root123, root124`.... Thus, this feature should be used as part of a comprehensive analysis in conjunction with other features and metrics.

Total Dictionary Overlap The total dictionary overlap $o_t(x, L)$ can be used to estimate the threat-level of an attacker. Figure 4.8 shows the CDF for our dataset. It is clearly visible, that the overall overlap is quite low with a maximum value of roughly 22%. This is expected, as the metric is calculated as the mean overlap between all other IP addresses in the dataset and large dictionaries score lower values for this metric when compared with smaller ones, because the overlap is computed for all other dictionaries and small dictionary can only contain a fraction of a larger dictionary. Given that most dictionaries are small ($\leq$100), the small values for $o_t(x, L)$ are expected.

As the effective range for the total dictionary overlap is small (in our dataset), differences between attackers are more significant than in other metrics. Especially very low values in the range (0,0.01) are interesting, because this indicates either a new dictionary that not many attackers use yet or a targeted attack that uses potentially valid credentials that consequently do not appear in any other dictionaries. The metric can therefore be used as a prioritization metric when applied to either single IP addresses or clusters. Additionally, a threat hunter could calculate the dictionary overlap for any other combination of selected attackers or clusters to further analyze potential collusion or similarity in behavior.

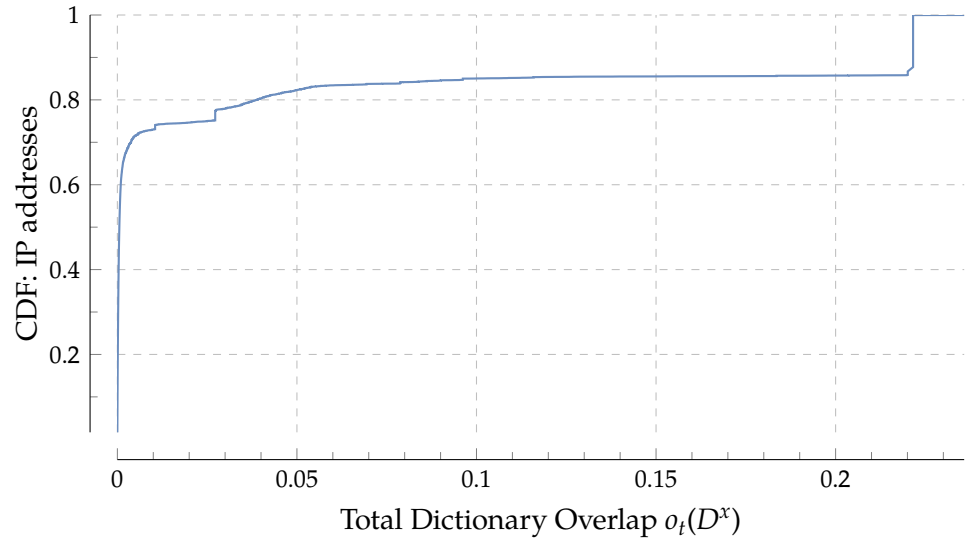


FIGURE 4.8: CDF: Total Dictionary Overlap $o_t(D^x)$

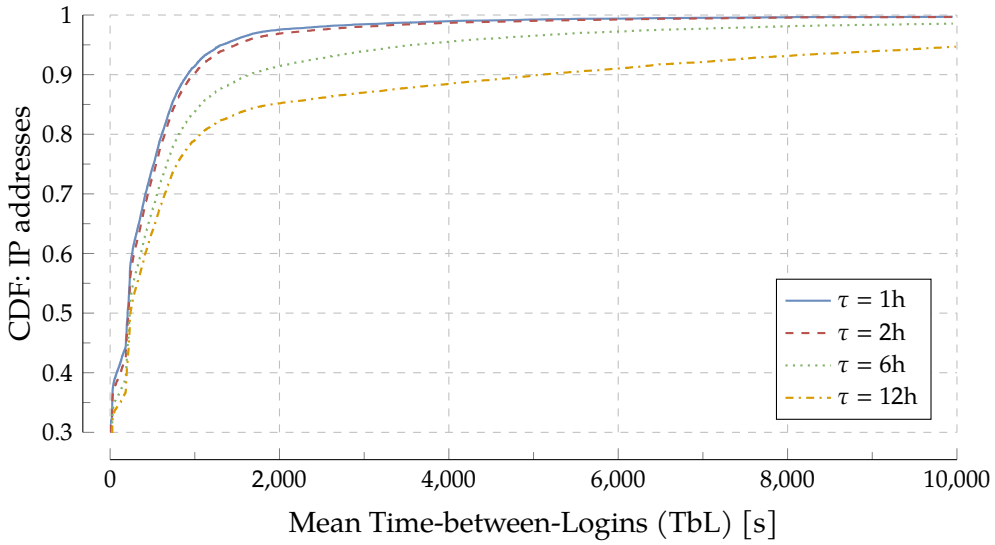


FIGURE 4.9: Influence of session timeout τ on Time-between-Logins (TbL)

Time-between-Sessions/Logins When considering brute-force sessions, it is important how τ is chosen. As described in Section 4.2.4, τ describes the session timeout between two login attempts. Intervals larger than τ start a new session.

Unfortunately, it is not possible to recommend an optimal value for this parameter, as there is not a single metric this decision can be based on. Figure 4.9 shows the influence of τ on TbL while Figure 4.10 covers TbS. The mean TbL $\delta_l(S^{x\tau})$ increases for larger values of τ. In the CDF this results in a lower curve as more IP addresses exhibit larger values. This is expected, as the maximum timeout inside a session is increasing and thus the mean. However, we can see that all graphs exhibit the same trends shifted on the y-axis. For TbS the impact is even less severe and all four graphs overlap nearly completely. This is somewhat expected as this results indicates that the TbS is significantly larger than τ and thus not influenced by it.

Ultimately, we recommend setting τ to a relatively low value such as 1 hour, as it

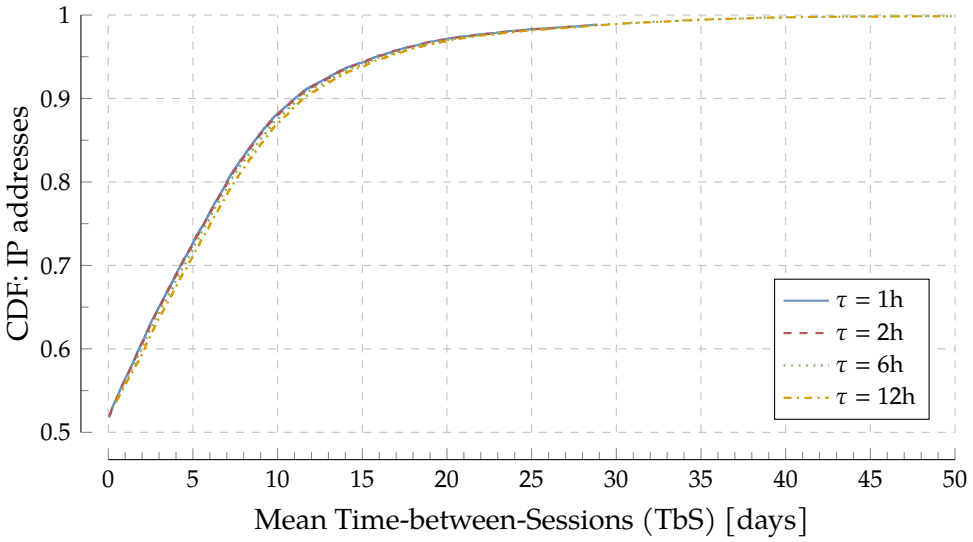


FIGURE 4.10: Influence of session timeout τ on Time-between-Sessions (TbS)

has no impact on the runtime of the algorithm prevents the merging of sessions as described in Section 4.2.4. This allows for more fine-grained session metrics. The parameter has massive impact on the single case but negligible impact on the means of the metrics. However, the risk of merged sessions for larger values of τ outweighs the benefits of fewer sessions that need to be processed for computation of further metrics. Consequently, all following analyses were conducted with $\tau = 1$ hour.

Dictionary Entropy The dictionary entropy E^x offers a rough indication about how the attempted passwords change in relation to another over time. A low mean value or average change between attempts indicates a credential sequence that is changing slowly, a property that is often explained by either ordered wordlists (similar initial characters) or multiple number suffixes. Both might indicate a low-threat attacker that is non-targeted and attacks a broad set of targets. A higher value or average change thus hints a more sophisticated attacker that either uses a tailored credential list or at least tries credentials at random to divert the analyst. As we do not expect many high-threat attacks in our dataset, the expected results for our dataset should be low to medium. Figure 4.11 shows a CDF for `mean`, `maximum` and `minimum` values of E^x for our dataset.

4.2.6.2 Clustering Results

A single metric is not sufficient to capture an attacker's behavior. While the analysis of single metrics gives a good indication of the general threat landscape the organization faces, they show their true potential when used in combination. In this section, we show clustering on multiple metrics can reveal common behavior shared across IP addresses as well as help prioritizing certain attackers that appear as outliers in the clustering process. All clustering in this section was performed via DBSCAN [EKX96] with varying parameters and data was normalized to unit variance before clustering. We performed most of the metric calculation in Rust and supplementary calculation and clustering in Python using *scikit-learn* [Ped+11].

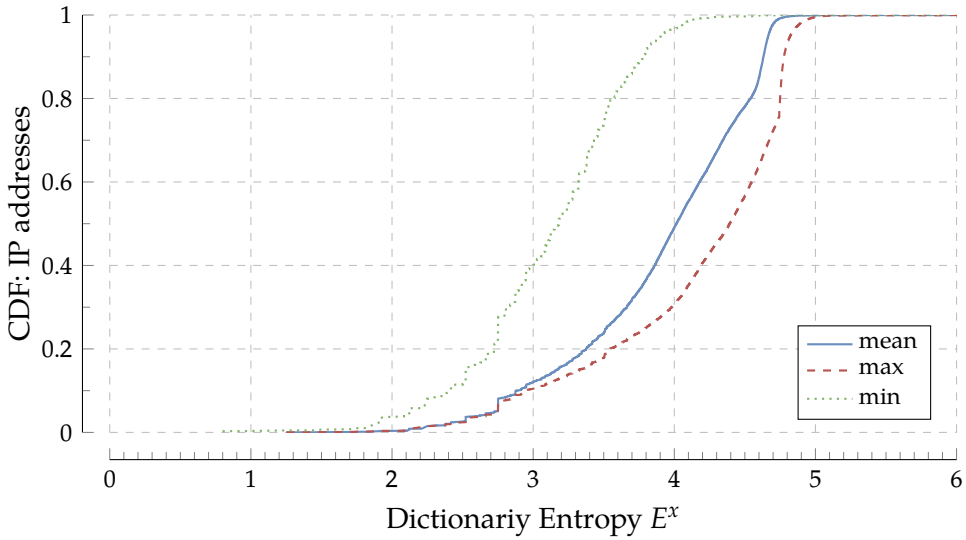


FIGURE 4.11: CDF: Dictionary Entropy E^x

Dictionary Size and Total Dictionary Overlap Clustering based on Dictionary Size and Total Dictionary Overlap together is useful because it highlight certain combinations that are especially interesting. For $|D^x|$ both low and high values can be interesting, as a small dictionary might indicate a very targeted attack while a large dictionary is useful to learn about potential new wordlists. The overlap is usually most interesting for low values as this shows, that the credentials are not usually found in other dictionaries. We used these two metrics to cluster the dataset via DBSCAN (ϵ = 0.15/`min_points` =2). Figure 4.12 shows the color-coded clusters without outliers (unclustered IP addresses).

We obtained 18 clusters and 52 outliers this way. The graph also shows that two clusters are by far the largest (shaded in dark red around 0 on the x-axis). These two clusters with 12 170 and 2 055 IP addresses respectively account for 95% of all IP addresses. All other clusters range between 2 and 42 IP addresses. These amounts are much more manageable for manual analysis than the full dataset.

After obtaining these clusters, the analyst would have to do manual follow-up inspection of the clusters and outliers to reveal potential cooperation or prioritize defensive measures. While our own analysis is not exhaustive we found that the two /24-subnets, that we already found suspicious during initial manual analysis, were largely clustered together accordingly. 218.92.0.0/24 and 58.242.82.0/24 both contain attackers with very similar dictionaries and partially directly following IP addresses. While this shared behavior is easily detectable by basic features such as subnet and dictionary size, these results show that clustering in this case indeed found potentially colluding attackers and is thus useful to correlate IP addresses.

Mean Session Duration and Mean Time-between-Logins (TbL) Another interesting combination of metrics is (mean) session duration and (mean) TbL. The session duration is a potential indicator of the configuration in the attacking tools or scripts. Login attempts are often interleaved with breaks to avoid detection. We clustered the dataset by these two metrics via DBSCAN (ϵ = 0.15/`min_points` = 2) and obtained 66 clusters and 152 outliers. These clusters are visualized in Figure 4.13.

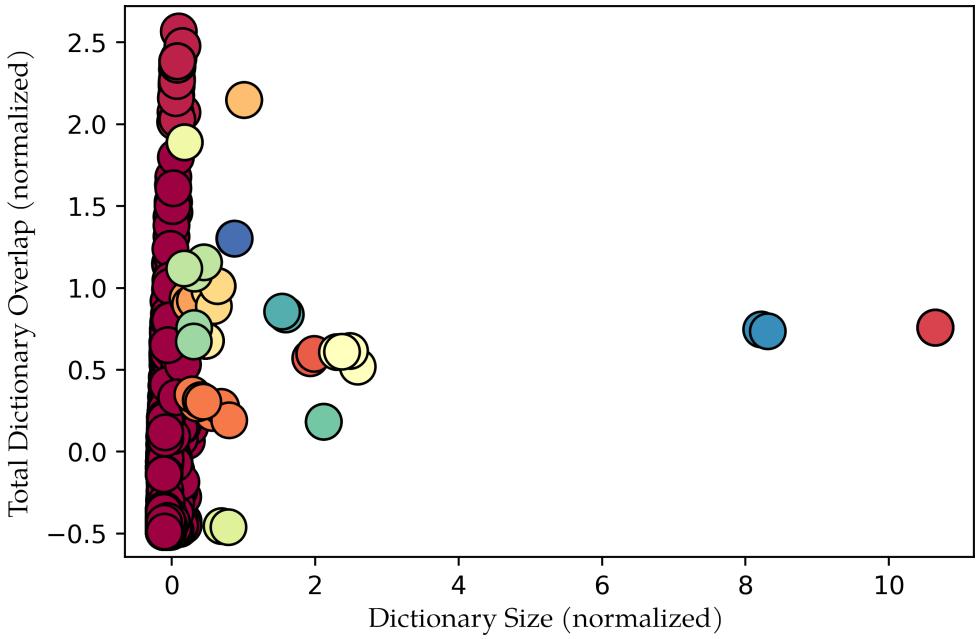


FIGURE 4.12: DBSCAN: Dictionary Size and Total Dictionary Overlap (no outliers)

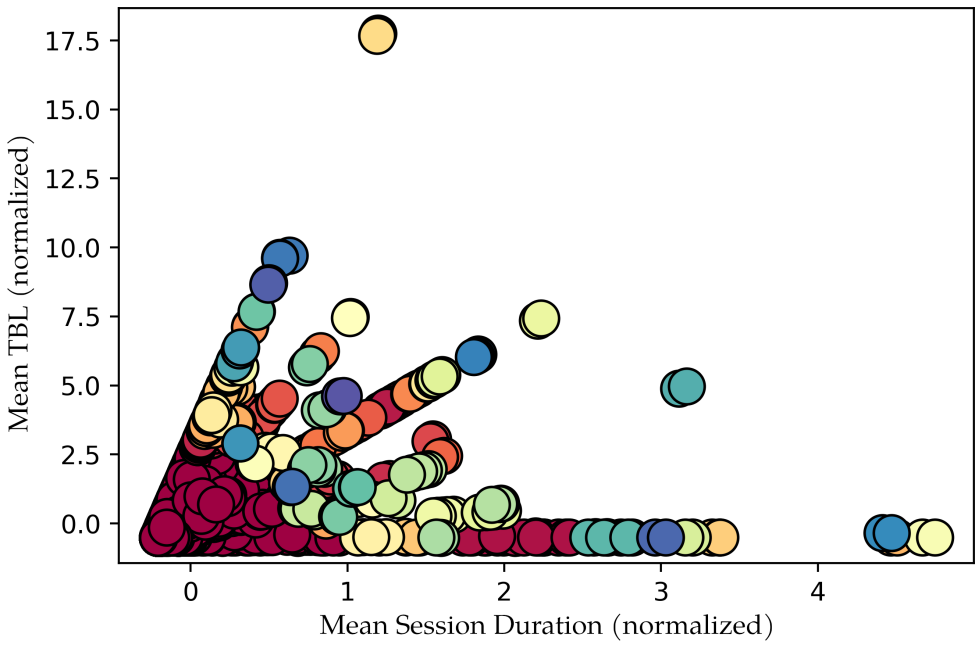


FIGURE 4.13: DBSCAN: Mean Session Duration and Mean TbL (no outliers)

Cluster	IP addresses	E^x	$v_l(L, x)$
1	12 170	**3.7717**	0.1409
2	2 055	4.6324	0.9706
3	42	4.6173	130.4454
4	2	4.5007	14.2738
5	8	4.5246	3.4849
6	3	4.7320	2.0620
7	3	4.7680	**996.1524**
8	16	4.7132	3.6055
9	10	*4.6617	*2.8451
10	4	4.6419	19.4871
11	11	4.7058	1.4431
12	2	4.6441	4.6825
13	4	4.6965	3.6129
14	2	4.6102	2.9367
15	3	4.2565	45.2264
16	2	4.8489	12.7381
17	2	4.6735	85.4704
18	2	4.5982	11.1378

Table 4.4: Results: Priorization for clusters

While both the cluster and outlier count is higher then for the previous metric combination, the results still offer some interesting insights. The visualization clearly shows the largest cluster containing a large amount of attackers that score relatively low in both metrics. Besides that, several lines of clusters are visible that are very linear with some small clusters further away. These lines indicate a linear relation between the two metrics as the quotient between the session duration and TbL remains constant on a line. This might be the result of attackers using the same timeout between login attempts but different number of maximum attempts before a larger break is inserted, i. e., a new session is recorded. Unfortunately we could not achieve clustering of separate lines with DBSCAN, so new algorithms could be evaluated for this metric combination. Nonetheless, an analyst could not examine clusters on the same line in conjunction and try to find other common behavior such as similar patterns in the credential sequence indicating similar tool usage. Also the relatively high amount of outliers (152) would have to be evaluated further.

4.2.6.3 Metric-based Prioritization

Features and metrics can also be used to prioritize either single IP addresses or clusters of IP addresses obtained as described in the previous section. The selection of the concrete metric is dependent on the goal the analyst aims to achieve. While most metrics focus on estimating attacker sophistication (and therefore threat-level), some corner cases might have other goals such as selecting a large cluster of low-threat attackers that can be dealt with easily. In this section we calculate selected metrics for the clusters obtained via the metric combination "Dictionary Size and Total Dictionary Overlap" and describe the results.

As mentioned earlier we obtained 18 clusters and 52 outliers. We chose two metrics to prioritize the cluster for further analysis. Both metrics serve different purposes and a

threat hunter would have to carefully select which metrics to follow up first as this is extremely dependent on the scenario. First, we chose (mean) Dictionary Entropy E^x as a general indicator of the attacker sophistication. Second, we selected the (mean) Login Frequency $\nu_l(L, x)$ under the assumption that an APT attacker would try to perform as few login attempts per day as possible to remain undetected. Table 4.4 contains the results for these two metrics as well as the number of IP addresses in the clusters.

As the table shows E^x is significantly lower for the largest cluster indicating that the password used there are less complex than in the other clusters. However, it proves to be not as good for further prioritization as the other values are very close to each other. Combined with the size of the cluster however, it might still prove useful and is still less complex than looking at the dictionaries manually. $\nu_t(L, x)$ shows some interesting differences between the clusters. Cluster 7 scores by far the highest value while many other clusters contain attackers that performed less login attempts per day. Depending on the scenario the threat hunter could not investigate either cluster 7 or cluster 11 when expecting APT attackers.

We also discovered a very strong indicator for collusion between IP addresses in cluster 9. This cluster contains ten IP addresses from distinct subnets that look benign on the first look. However, they scored exactly the same values for both E^x and $\nu_l(L, x)$ with a standard deviation of 0.0 among them. This strongly indicates cooperation or at least usage of the same scripts and would have been difficult to detect otherwise. The threat hunter could now follow up an investigate if any other IP addresses from other clusters scored the same for these two metrics. In any case this result shows, that clustering based on our set of metrics can correlate otherwise unrelated IP addresses and thus support the threat hunter effectively.

In summary, our evaluation indicates that feature- and metric-based analysis of brute-force login attempts can provide valuable insights into their behavior and help to better assess the resulting part of the external threat landscape. The combination of the previously proposed feature and our two newly proposed metrics can be used to (i) cluster login attempts from unrelated IP addresses to reveal potential collusion and (ii) prioritize both single attackers or clusters of attackers. While ground truth for real-world brute-force logins is not available, the analysis of our dataset obtained through our Honeygrove [Hon+22] honeypot revealed interesting behavioral patterns and clustering yielded groups of unrelated IP addresses that exhibited highly similar behavior and are likely colluding.

4.2.6.4 Requirement Comparison

Based on the description and evaluation results, we can now compare our approach for metric-based characterization of brute-force attackers with our requirements formulated in Section 3.1. *R1: Accuracy* is difficult to assess as there is no ground truth available for malicious brute-force login attempts. However, we obtained solid indicators of collusion and thus mark it as partially fulfilled. *R2: Explainability* is fully met as the set of metrics (and especially our two novel metrics) allow for deep insights into attacker behavior that support SOC analysts in their investigations. As our approach uses common features that are usually captured by security infrastructure, *R3: Low overhead* is also checked. While not explicitly measures, we rate *R4: Scalability* as fulfilled as our calculations are not computationally intensive and easy to parallelize, e. g., by dividing the incoming attempts per IP and distributing these streams to

different machines. Both *R5: Security* and *R6: Privacy* are not directly applicable as our approach only processes data provided by "publicly" over the Internet by attackers. The only potential impact would be the credentials of legitimate traffic, however an organization could leverage an approach as presented by Herley and Schechter [HS19] to distinguish between legitimate and malicious attempts before applying our metrics. Lastly, *R7: Deploy- & Maintainability* is fully met as the required information is lightweight (and thus usually available in enterprise contexts) and our approach does not rely on other components that would need to be deployed and maintained.

4.2.7 Summary

This section introduced an approach to characterize behavior of brute-force attackers using a collection of established and novel metrics to answer our formulated research question **RQ2**:

RQ2 *How and to what extend can brute-force attempts on externally-accessible services be categorized and screened for potential APT reconnaissance activity?*

We described how the features and metrics of login attempts can be used in conjunction or in isolation to cluster and prioritize IP addresses for analysis. The set of metrics obtained from our brief literature survey is enhanced with two novel concepts that enable detailed insights into attacker timing behavior of connected attempts as well as entropy of their dictionary. In our evaluation, based on real-world login attempts collected by our honeypot Honeygrove [Hon+22], we show examples how single metrics or combinations of them can be used to find clusters of similar behavior. The resulting clusters can then be further analyzed by several metric combinations, e. g., to estimate their threat level. The resulting information can be used during manual investigation by SOC analysts: while large clusters are probably the result of automated scanners or similar attacks, unclustered or otherwise abnormal login attempts might be results of adversarial reconnaissance activity, i. e., the first stage in a future APT campaign. Overall, our evaluation demonstrated how our approach can help to estimate attacker sophistication and thus potentially help to identify APT reconnaissance activity and answering **RQ2**. The resulting information can act as additional indicators for attacker attribution among other intelligence.

4.3 Summary

Security monitoring and the resulting visibility is the foundation for any successful intrusion detection and especially important for stealthy APT attacks that aim to stay undetected. This chapter introduced two core contributions of this thesis that provide better visibility into the threat landscape of brute-force login attempts as well as into payloads of TLS-encrypted communication. Together, these two approaches improve the overall internal and external visibility that can leveraged for security analytics such as the detection approaches presented in the next chapter.

Section 4.1 described an approach for passive TLS decryption for NMSs based on key material that is shared by cooperative endhosts. This significantly improves upon the predominant approach of TLS interception in enterprise networks, i. e., deployment of MitM proxies that intercept all TLS connections at the network edge and forward cleartext payloads to a NMS. In contrast, our approach (*i*) preserves the end-to-end integrity guarantees of TLS, (*ii*) conceptually supports selective decryption and thus

allows users to retain privacy in some scenarios, and (*iii*) reduces the threat surface slightly by shifting the access to cleartext payloads from active proxies to passive NMSs. Our results indicate that our prototype implementation is feasible for real-world scenarios with decent overhead imposed by the encryption and minimal traffic buffering required in front of the decrypting NMS. Furthermore, an extended version of our prototype has been merged into Zeek mainline and is included by default starting from version v5.0.0 [AW22].

The approach presented in Section 4.2 aims to characterize behavior of brute-force attackers. The combination of established metrics from literature as well as two novel metrics (brute-force sessions and dictionary entropy) can be used to both cluster and prioritize IP addresses for analysis. Our evaluation based on our real-world dataset BFL2020 [WF20a] obtained through Honeygrove [Hon+22], showed that strong indicators for collusion between otherwise unrelated IP addresses could be obtained and which metric combinations were promising for prioritization. While the results are highly dependent on the data and are likely to differ for other scenarios, the results are promising and indicate that our metrics can support analysis of brute-force login attempts.

In summary, this chapter introduced two methods to improve security monitoring for both internal and external visibility. Approaches in this area are essential as they provide the foundation for higher-level correlation and detection. Both contributions can be used in parallel or individually and acts as enablers for other algorithms, e. g., detection scripts in Zeek [Zee22] that can leverage the restored visibility into TLS payloads for deep packet inspection on application layer. The next chapter proceeds to the area of security analytics and introduces one stage-specific approach to identify hosts affected by lateral movement of an APT campaign, a concept to restore explainability to graph-based APT campaign detection approaches, and a whole-campaign reconstruction approach based on the Kill Chain State Machine (KCSM) that generates compact visual representations for SOC analysts. These five approaches in total represent the main contribution of this thesis.

5 Approaches for APT Detection

This chapter contains three contributions that can be used to detect advanced persistent threats (APTs) detection and thus offer different approaches to effectively support security operations center (SOC) analysts in their investigations. The first contribution has been published at ARES 2019 [Wil+19b] and describes an algorithm to reconstruct lateral movement activity in a forensic setting based on a formal model of security-relevant host properties. We propose an abstract algorithm as well as two concrete implementations that ingest an incomplete set of alerts/indicators of compromise (IoCs) and generate a set of hosts that were likely compromised during lateral movement. The second contribution was submitted to IEEE ICC 2023 [WWF23] and describes an approach to explain classification results of anomaly detection-based APT detection systems. Our approach leverages a variation of permutation importance (an established technique from explainable artificial intelligence (XAI)) and can be applied to any graph-based anomaly detection approach across application domains. The third contribution has been published at CYSARM 2021 (co-located with ACM CCS 2021) [Wil+21] and describes an approach to reconstruct APT campaign activity. The foundation of this process is the Kill Chain State Machine (KCSM), a formalization of the comprehensive unified kill chain (UKC) [Pol21], that describes pre- and post-conditions of APT stages and thus can be used to link alerts based on it.

Each of these three approaches improves a relevant part of the overall APT protection process. The lateral movement reconstruction approach generates stage-specific alerts that can be used in higher-level detection algorithms. For enterprise scenarios, which already leverage system provenance-based detection systems, our explainability approach helps to establish attack context to support SOC analysts. Both types of alerts, stage-specific and anomaly-based, can then be used as part of our approach for APT campaign reconstruction via KCSM. As a result, this thesis offers solutions to singular problems along the APT detection process (the first two contributions of this chapter) as well as a higher-level reconstruction approach that can ingest alerts from several detection systems (both from literature and the proposals in this thesis).

5.1 Reconstruction of Attacker Lateral Movement

This section presents an approach to reconstruct lateral movement of different classes of attackers based on a notion of *host criticality*. This helps to estimate the impact of an APT campaign and thus speeds up incident response. Overall, this contribution aims to answer research question **RQ3** as introduced in Section 1.1:

RQ3 *How can APT mitigation be supported by combining both alerts and security-relevant host information, e. g., to estimate impacts of the attack?*

Criticality, a host attribute of the formal model our approach is based on, emphasizes hosts that are both highly relevant for the overall network function and also vulnerable, making them a prime target for APT attackers. We propose an abstract algorithm

to reconstruct potential lateral movement activity from an incomplete alert set (as commonly present in forensic analysis) as well as two concrete implementations based on (*i*) random walks and (*ii*) k-shortest paths. The resulting ordered list helps analysts to prioritize hosts for analysis, thus speeding up incident response time which is especially critical in APT scenarios. This contribution is based on preliminary work from a Bachelor's thesis [Fra17]. Furthermore, this section shares material with the corresponding conference publication [Wil+19a] that significantly revises and extends the approach developed in the Bachelor's thesis.

5.1.1 Motivation and Objectives

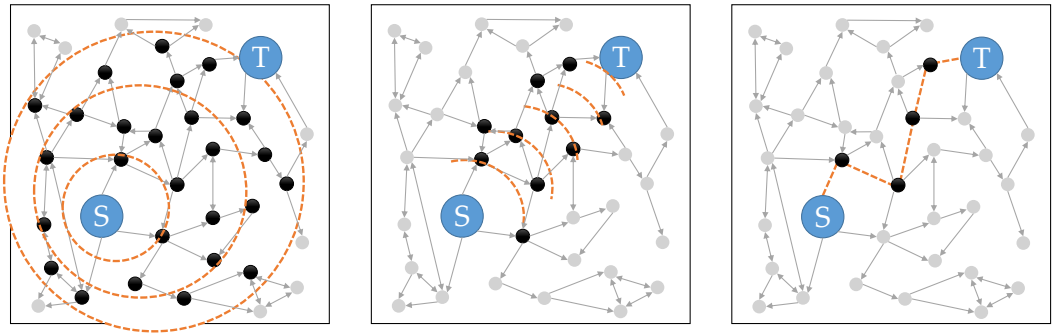
Lateral movement is one of the key phases of complex multi-stage attacks such as APT campaigns. Attackers continuously expand their reach in the target network until they either reach their campaign goals or are detected and thus deterred. After detection, it is thus critical to reveal which hosts were infected as part of the campaign. Knowing this set of infected hosts is important to both understand the campaign and attacker behavior and for cleanup measures during incident response. However, as detection only yields an incomplete set of IoCs or alerts, reconstruction of the attacker's lateral movement to obtain the set of infected hosts remains difficult. In the worst case, security experts have to assume the whole network is compromised and inspect and cleanup all hosts, a very time-consuming task that is detrimental both financially and reputational as public services might not be available during incident response. To address this problem, this contribution aims to achieve the following *objectives* while also fulfilling our overall requirements formulated in Section 3.1:

- Design a model to capture both auxiliary data of the underlying network infrastructure as well as attacker behavior during lateral movement.
- Devise an algorithm to systematically reconstruct lateral movement activity based on a given network topology and an incomplete set of IoCs.
- Evaluate the approach based on data that approximates real-world lateral movement activity as close as possible.

The remainder of this section is structured as follows: Section 5.1.2 introduces the formal models for network and attacker the approach is based on. Section 5.1.3 describes the high-level algorithm to reconstruct attacker lateral movement. Section 5.1.4 details two concrete variants of the proposed algorithm based on (*i*) k-shortest paths and (*ii*) biased random walks. In Section 5.1.5 we evaluate our approach based on simulated alert data and compare it with the requirements formulated in Section 3.1.

5.1.2 Formal Model

Detecting APT attacks, especially with respect to the attacker's lateral movement, requires a thorough understanding of how an attacker potentially spreads within the network in question. Therefore, we propose formal models for both the network and different classes of attackers. This is the prerequisite to quantify attacker interest in different hosts and to reconstruct their lateral movement through the network.



(A) Basic (B) Directional (C) Insider

FIGURE 5.1: Lateral movement graphs for different attacker models. Attackers move from start host S towards target host T. Orange-dashed represents the area of the lateral movement graph, black hosts have been compromised, gray hosts remain untouched.

5.1.2.1 Network Model

The network model is based on a directed reachability graph $G_R = (V, E)$ as this matches the possibilities an attacker has during lateral movement. Hosts are modeled as vertices V while edges E represent connectivity between them. The model uses a simplified definition of connectivity that does not consider advanced routing configurations. That is, if any connection can be established from host u to host v, G_R contains the edge (u, v). This is in contrast to other approaches which usually consider connectivity based on port/service.

Furthermore, we define two vertex attributes, *Importance* and *Vulnerability*, that represent properties of the corresponding host that are visible to an attacker. This includes security properties and appliances such as network-based intrusion detection system (IDS), host sensors, and other endpoint protection solutions.

The *Importance* $I(v) \in (0, 1]$ represents the value of the host for the network. Hosts which run core services such as application and database servers are therefore rated high while regular workstations typically receive a low value. The lower bound explicitly excludes 0 as any compromised host represents a security risk and therefore should not be ignored. $I(v)$ can be semi-automatically derived from installed software when applying the approach to an existing network. Each software is rated with a certain value which ultimately results in a cumulative *Importance* score for each host. An attacker can estimate $I(v)$ by scanning a host for open ports or monitoring network traffic to deduce which services are running on the host.

The *Vulnerability* $V(v) \in (0, 1]$ represents the difficulty for an attacker to compromise the host. This attribute is influenced by a variety of factors such as patch level of the operating system and installed software, known vulnerabilities matching the system configuration, and security policies implemented in the network. The lower bound explicitly excludes 0 to account for zero-day exploits as they can be used to compromise even the most secure hosts. Similarly to $I(v)$, $V(v)$ can be derived semi-automatically from the installed (security) applications on the respective hosts and automated vulnerability scans such as Nessus [Ten22] or other CVSS-based [For15] scanners. The *Vulnerability* also allows to account for the human factor in IT-security by increasing the value for all hosts that are regularly used by employees, e. g., workstations or even employee-owned smart devices. An attacker can also estimate $V(v)$ by

deducing software versions from banners, raw network packets or other fingerprinting techniques.

$$C(v) = I(v) \cdot V(v) \tag{5.1}$$

Based on these attributes we can assign a *Criticality* $C(v) \in (0,1]$ value to each host in the network as given in Equation (5.1). This value represents a prioritization score of the attacker. As these scores can be compared among the hosts, they are used in our attacker model to approximate which hosts an attacker most likely tried to compromise in each lateral movement step. We chose the product over a linear combination of $I(v)$ and $V(v)$ to highlight hosts which are both *important* and *vulnerable* as especially *critical*. It is therefore important to compute accurate values for both $I(v)$ and $V(v)$ when applying our approach to a real network. While a certain mismatch between the knowledge base of the security administrator and the attacker can be expected (and addressed in our approach), a more accurate estimation is expected to boost our algorithm's performance.

5.1.2.2 Attacker model

Based on the formal network model we can define three attacker models to capture the distinct behavior of different classes of attackers. We differentiate between *basic attackers*, *directional attackers* and *insider attackers*. These three attacker classes differ mainly in their knowledge about the target network.

In our model, all attackers start from a random *vulnerable* host in the network and target an *important* host, e. g., the production database or a crucial gateway system. This closely resembles real attack scenarios, where the initial point of compromise is generally hard to influence by the attacker, but most often is one of the more *vulnerable* hosts. The attackers then move laterally through the network and aim to find the best path to the target host. The *Criticality* $C(v)$ hereby models the cost function an attacker tries to maximize when deciding which host to compromise next. Depending on the attacker's knowledge this results in either a straight path or tree like movement to find the target host.

While these models describe an "idealized" attacker of the respective class, a real intruder will probably deviate from the expected path. This can either be explained by the different knowledge bases of attacker and security administrator mentioned in Section 5.1.2.1 or additional reasons specific to each class of attackers.

Basic attackers These are intruders with strong technical skill that lack detailed information about the target network. To find the best path to the target host, basic attackers have to gradually explore the network while prioritizing hosts with high *Criticality*. Therefore, they generally compromise large parts of the network before reaching the target host. Their lateral movement forms a *broad tree-like graph* that is visualized in Figure 5.1a. Reason for *deviation* from the best path, besides the aforementioned mismatch of knowledge, could be exploits (possibly even zero-days) that enable the attacker to compromise an otherwise secure host.

The behavior of basic attackers can be modeled by Dijkstra's algorithm [Dij59]. The link distance function $d(u, v)$ can be modeled by taking the inverse of the *Criticality* of the destination node as given in Equation (5.2). However, in contrast to most

applications of Dijkstra's algorithm, the lateral movement consists of all nodes that were explored instead of the shortest path only. This represents the explorative nature of the lateral movement of basic attackers.

$$d(u, v) = \frac{1}{C(v)} \tag{5.2}$$

Directional attackers Intruders with limited, structural information on the network are referred to as directional attackers. They also have to gradually explore the network to find the best path to the target node, but can use their knowledge to make better decisions which host to compromise next. An attacker from this class might know in which subnet the target host is located as well as how subnets are connected. They can then use this information to disregard a host with otherwise high *Criticality* and instead compromise another host that is reachable in different subnet close to the target host. Thus, their lateral movement graph is usually a *medium-sized tree* as multiple branches have to be explored to find the optimal path to the target. Figure 5.1b shows an example graph for this attacker class.

The behavior of directional attackers can be modeled by the A^* algorithm [HNR68]. The heuristic hereby represents the attacker knowledge and limits the search space the attacker has to explore. It is therefore highly important to choose an appropriate function that closely approximates real behavior.

We chose an euclidean heuristic based on node coordinates assigned by a multidimensional scaling algorithm. Each node gets embedded in a two dimensional plane based on their connecting edges. This results in (x, y) coordinates for each node which represent an abstract location of the host in the network. This could represent a particular subnet or even real geographic coordinates. The heuristic then just calculates the euclidean distance between the target host v and the current host u (as shown in Equation (5.3)). The lateral movement graph consists of all nodes that were explored by the A* algorithm.

$$H(u, v) = \sqrt{(X(v) - X(u))^2 + (Y(v) - Y(u))^2} \tag{5.3}$$

Directional attackers might *deviate* from the best path for the same reasons as basic attackers, but especially if the heuristic does not accurately model the knowledge of the real attacker.

Insider attackers We refer to intruders that know the complete network topology, as well as the security properties of each individual host, as insider attackers. Their total knowledge enables them to traverse the network directly from the initial point of compromise to the target host in the most efficient way possible. As a result their lateral movement graph is usually a *simple path* through the network (visualized in Figure 5.1c). However, insider attackers might still *deviate* from the optimal path for various reasons such as

- (zero-day) exploits (which by definition cannot be considered when calculating host vulnerability),
- legitimate credentials,

- when attackers try to avoid security appliances, or
- if an attacker explicitly tries to cover their tracks by, e. g., partially behaving like an attacker of a different class.

This class of attacker can be modeled by any shortest path algorithm that accepts custom edge costs. In the concrete case of directed reachability graphs, Dijkstra's algorithm is again sufficient and the resulting shortest path represents the attacker's lateral movement. In the next section we will present algorithms to efficiently approximate the lateral movement of these three attacker classes from an incomplete set of IoCs.

5.1.3 Approach Overview

After an APT attack has been discovered, it is essential to efficiently identify which hosts were compromised during the attacker's extended presence in the network to recover quickly and harden the network against future attacks. We present an approach to compute a candidate set of hosts which most likely have been compromised based on the IoCs that could be obtained from the attack.

We assume preprocessed IoCs that have been verified and attributed to the APT attack in question. In particular, we assume that there are no false positives in the IoC set. For alerts from IDSs this usually involves some kind of aggregation or correlation such as presented in [HF18]. For simplicity, we also assume a IoC was detected on the first and last node in the lateral movement graph, i. e., entry point and target host are known.

The considered problem can then be stated as follows: Given a weighted reachability graph G_R that has been annotated according to our network model (see: Section 5.1.2.1) and a set of strictly ordered IoCs I^*, we aim to identify the target set of hosts the attacker compromised on their way to the target host V_t. As the lateral movement cannot be fully reconstructed without human validation, the algorithm returns an approximated candidate set of compromised hosts V_c. Additionally, V_c should be ordered, such that the nodes that are most likely compromised, can be inspected first.

Input : G_R, I^*, τ
Output: V_c

```
1  i ← 0;
2  V_c ← ∅;
3  while |V_c| < τ do
4  |   foreach (u, v) ∈ I* do
5  |   |   nodes ← GetNodes(G_R, u, v, i);
6  |   |   Update(V_c, nodes);
7  |   end
8  |   i ← i + 1
9  end
10 return V_c
```

Algorithm 1: Lateral movement reconstruction from incomplete alert sets

The abstract algorithm we designed to address this problem is shown in Algorithm 1. The basic idea is to approximate V_t by repeatedly iterating the IoC pairs in I^* and obtaining nodes between them, until the result set is large enough ($|V_c| < \tau$). The

candidate set V_c then contains an approximation of the unknown target set of compromised nodes V_t. The desired size of V_c is given by the parameter $\tau \in (0,1]$ and expressed as a percentage of all nodes in the graph $|G_R|$. *Note*: The algorithm specifies neither `GetNodes` nor `Update` and thus only provides a blueprint how lateral movement can be reconstructed. However, any implementations should ensure that $V_c$is an ordered set of nodes, to use this order for prioritization of hosts during manual security analysis.

5.1.4 Implementation Variants

We implemented two variants of the abstract algorithm proposed in the previous section to evaluate our idea. One variant based on *k-shortest paths* and one based on *biased random walks*. This section motivates and explains both variants in more detail.

5.1.4.1 K-shortest Paths

Shortest paths are the intuitive choice to find nodes connecting two IoCs. The first variant thus leverages a k-shortest paths algorithm with the distance function based on "inversed Criticality" that is also used as the basis for our attacker models in Section 5.1.2.2. As the algorithm repeatedly calls `GetNodes` to obtain increasing numbers of nodes, we need an efficient implementation to compute multiple shortest paths between u and v. Yen's k-shortest paths algorithm [Yen71] provides this, as it is able to compute multiple shortest paths with increasing length while keeping state between iterations to avoid costly recomputations. The resulting concrete algorithm is shown in Algorithm 2.

Input : G_R, I^*, τ
Output: V_c

```
1  i ← 0;
2  V_c ← ∅;
3  while |V_c| < τ · |G_R| do
4  |   foreach (u, v) ∈ I* do
5  |   |   nodes ← K-Shortest-Path(G_R, u, v, i);
6  |   |   Update(V_c, nodes);
7  |   end
8  |   i ← i + 1
9  end
10 return V_c
```

Algorithm 2: Lateral movement reconstruction via k-shortest paths

Compared to the abstract algorithm, `GetNodes` is specialized to `K-Shortest-Paths` with the iteration parameter i used as the k parameter. The result set V_c is an append-only list of nodes and thus directly fulfills our order requirement. Nodes are added in the order they are discovered by the k-shortest paths algorithm. The `Update` function then appends new nodes found in the current iteration to the V_c by first checking for duplicates.

The first variant represents an intuitive approach for lateral movement reconstruction based on the underlying formal model. The k-shortest path function should perform decent for all three attacker models as the distance function is identical. However,

this assumes knowledge parity between attacker and defender which would result in identical scores for $I(v)$, $V(v)$, and $C(v)$. Most likely, this will not generally hold for real-world scenarios. Additionally, the detection performance will likely decline for *deviating attackers* as deviations are unlikely to be on the shortest path. This is also the motivation for the next variant.

5.1.4.2 Biased Random Walks

The second variant is based on biased random walks. Compared to the previous approach, this algorithm should be more robust against attackers, that deviate from the proposed idealized attacker models. However, pure random walks without any other factors would completely disregard the attacker models and thus likely achieve a worse performance. To address this, we use *biased* random walks with a modified transition probability adapted to our attacker model. This second variant is shown in Algorithm 3. The outer loop is changed from the conditional `while` loop that checks $|V_c|$ to a `for` loop that is executed until the newly introduced `limit` parameter is reached. The inner loop still iterates over the IoC pairs as previously, however the `GetNodes` function is specialized to `RandomWalk`. As the random walks can only specify a start node, which is u in this case, the algorithm has to check if the target node v was hit before updating V_c with the hit counts of the walk. As the candidate set then also contains hit counts instead of just an ordered set of nodes, this variant has to `OrderAndTruncate` V_c to meet our goals specified in the abstract algorithm.

Input : G_R, I^*, τ, limit
Output: V_c

```
1  V_c ← ∅;
2  for i ← 0 to limit do
3  |  foreach (u, v) ∈ I* do
4  |  |  hitcounts ← RandomWalk(G_R, u, v);
5  |  |  if v in hitcounts then
6  |  |  |  Update(V_c, hitcounts);
7  |  |  end
8  |  end
9  end
10 return OrderAndTruncate(V_c, τ · |G_R|)
```

Algorithm 3: Lateral movement reconstruction via biased random walks

Each random walk simulates a single attacker which moves laterally through the network starting from u to v. `RandomWalk` returns after either v was hit or the random walk timed out, performing more than a predefined amount of maximum steps. The return value is a set of node hit counts that contains each node that was visited by the random walk. If v is in this set, the walk reached the target node and V_c gets `Updated` with the new hit counts. Instead of exiting the loop when enough results have been obtained, this algorithm takes a `limit` parameter. This parameter sets the number of random walks that are performed and thus directly influences the runtime of this variant. As each random walk roughly takes the same amount of time, the `limit` parameter offers fine-grained control over the runtime. In contrast the variant on k-shortest paths is harder to predict, as the runtime of each iteration takes more time as the shortest paths get longer.

After the loop was exited, V_c contains the cumulative hit counts for all nodes that were hit in at least one random walk. V_c is then ordered by hit counts descending and truncated to the desired size of $\tau \cdot |G_R|$. The result is an ordered list of the nodes that were hit the most in all successful random walks.

$$P_t(u, v) = \frac{1}{|S(u)|} \tag{5.4}$$

$$P'_t(u, v) = \frac{C(v)}{\sum_{s \in S(u)} C(s)} \tag{5.5}$$

In order to adapt the random walks to our attacker model, we modify the transition probability $P_t(u, v)$ to be *biased* towards $C(v)$. $P_t(u, v)$ describes how likely a random walk transitions from node u to node v. As shown in Equation (5.4) the conventional, *unbiased* function simply factors in the out-degree of u and assigns each possible successor $s \in S(u)$ the same probability. The biased transition function prioritizes nodes with higher *Criticality* as given in Equation (5.5). The transition probability is essentially a "relative *Criticality*" value normalized by the sum of *Criticalities* of all neighbors. This resembles the basic assumption of all three of our attacker models: Each attacker generally compromises hosts with higher values for $C(v)$ to find the best combination of *Vulnerability* and *Importance* out of all available hosts.

5.1.5 Evaluation

We implemented both variants of the algorithm in a Python-based prototype and evaluated the performance in different scenarios. In this section, we first describe the general experiment setup and datasets that were used followed by a discussion of the results.

5.1.5.1 Experimental Setup and Datasets

There are no datasets openly available for either real enterprise network topologies and few data sets containing IoC data. Large organizations fear the knowledge contained in these sets might enable future attacks. Therefore, we evaluated our approach on datasets consisting of two types of generated graphs as well as IoC sets from synthetic lateral movement.

We evaluated our algorithm on standard preferential attachment graphs as they are structurally similar to Internet or data center topologies [Gre+09; ALV08]. While these networks do not exactly match our targeted enterprise networks, the graphs should provide a decent approximation. The graphs were generated using the Barabási-Albert algorithm [AB02] with $m = 4$—i. e., 4 edges were added from a newly generated node to the existing graph.

Node Attributes After the graphs have been generated, they have to be prepared for the algorithm following our network model (see: Section 5.1.2.1). We assign *Importance* and *Vulnerability* attributes to all nodes to represent the security properties of the corresponding hosts. Both attributes are sampled from certain probability distributions to approximate real-world networks.

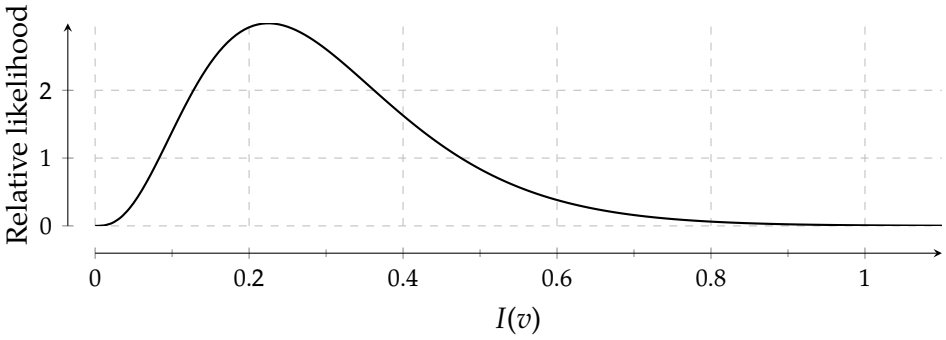


FIGURE 5.2: Probability distribution for $I(v)$ ($\alpha = 4$, $\beta = 0.075$)

In a typical enterprise network the *important* services (such as web servers, SSH gateways, or databases) are usually deployed on a small number of hosts, while the majority of the IP address space is occupied by regular workstations with relative low impact on the overall network function. Therefore, we sampled $I(v)$ from a *gamma distribution* with $\alpha = 4$ and $\beta = 0.075$ and limited the result to a maximum of 1 to fit our model. The corresponding probability density function is shown in Figure 5.2. The mode of this distribution is about 0.3 which matches an average workstation, while hosts with an attribute value of more than 0.6 are sampled less likely to account for typical server setups.

We assume that the *Vulnerability* of hosts is distributed inversely to their *Importance*. The majority of workstations are more likely to be outdated, uniquely configured or set up with a local administrative account controlled by the user. All of these factors increase $V(v)$ for these hosts compared to centrally provisioned server machines with automatic security updates. As a result, we sample $V(v)$ from $1-gamma(\alpha = 4$ and $\beta = 0.075)$ and limit the value to the required interval $(0,1]$.

As mentioned previously in Section 5.1.2.2, our implementation of the directional attacker requires node coordinates for the euclidean heuristic. We used *Gephi* [Gep22] with a plugin based on [Alg09] to perform multidimensional scaling and embed the nodes into the two-dimensional plane.

Attack generation After G_R has been prepared with node attributes, we generate synthetic attacks based on lateral movement along the three attacker classes. Moreover, we use a *deviation* parameter $\delta \in [0,1]$ that determines how much the synthetic attack deviates from the idealized attacker model. δ is applied on the graph by selectively setting the edge cost of a percentage of edges to $2^{63} - 1$, the maximum value of an 64-bit integer. For $\delta = 0.2$ this means that the cost for every fifth edge is set to this value, effectively removing it from the graph for the shortest path calculation. This simulates an attacker that deviates in 20% of cases, because 20% of edges in the graph will (most likely) not be on the shortest path to the target node.

The start and end nodes of each lateral movement graph are chosen based on our assumption that an attack starts on a *vulnerable* host and targets an *important* one (see: Section 5.1.2.2). To account for this we calculated all shortest paths between the nodes with the top 10% of *Vulnerability* $V(v)$ and the top 10% of *Importance* $I(v)$ according to the algorithm for the respective attacker class. We then choose the lateral movement graph from one of the 10% longest paths to have enough nodes to reconstruct later. For insider attackers that is just the path itself—for both other

	Basic attacker		Directional attacker		Insider attacker	
$\|V\|$	$\|V_c\|$	$\|I^*\|$	$\|V_c\|$	$\|I^*\|$	$\|V_c\|$	$\|I^*\|$
200	151	76	93	47	8	5
500	383	185	270	131	11	6

Table 5.1: Dataset Overview: Reconstruction of attacker lateral movement

attacker classes the lateral movement graph consists of all visited nodes. Lastly, the result stored as an ordered set of nodes.

Alert placement After the lateral movement has been generated, IoCs were randomly placed on the set of nodes. We introduce another node attribute called *Detectability* $D(v) \in [0,1]$ that indicates how likely a compromise is detected on the host. When applying the approach to a production network $D(v)$ can be derived from the installed security appliances in the network and on the hosts. For our synthetic networks, we assume a balanced security setup with some hosts protected by IDSs and some honeypots deployed in the network (which carry a high detectability). Therefore, we sample $D(v)$ from a *normal distribution* with $\mu = 0.5$ and $\sigma = 0.15$. To place the alerts a random number from $[0,1]$ is generated for each node in the lateral movement graph. If that number exceeds $D(v)$, an IoC is placed on the node. Additionally, the first and last node always receive an IoC—following our assumption that point of entry and target of the attack are known.

Datasets For our experiments, we generated two datasets differing mainly in the size of the preferential attachment graphs. Table 5.1 gives an overview across both datasets. Next to the number of nodes in G_R we also highlight the number of compromised nodes $|V_t|$ for each attacker class on average and the average number of alerts placed on these nodes $|I^*|$. The numbers give a rough overview about the dimensions of an attack from the respective attacker class. Basic attackers compromise about 75% of nodes on average while directional attacker reach the target node after compromising 46–50% respectively. Insider attackers only need to compromise 2–4% of all nodes as they can take the shortest path directly. Initially, all experiments were performed on both datasets but we saw that network size (at least between 200 and 500 nodes) had no significant impact on the results. Therefore, all graphs in the following section were obtained from the result set with $|V| = 200$ with 50 repetitions to reduce runtime.

5.1.5.2 Metrics

To evaluate our algorithm, we performed multiple experiments based on the setup described in the previous section. In the following, we treat the two algorithms as binary classifiers,i. e., the hosts that are contained in V_t are classified as compromised, while the remaining ones are classified as non-compromised. This is formalized in Equation (5.6). The *true positive rate* (*TPR*) is calculated by dividing the number of correctly labeled host by the total number of compromised hosts V_c.

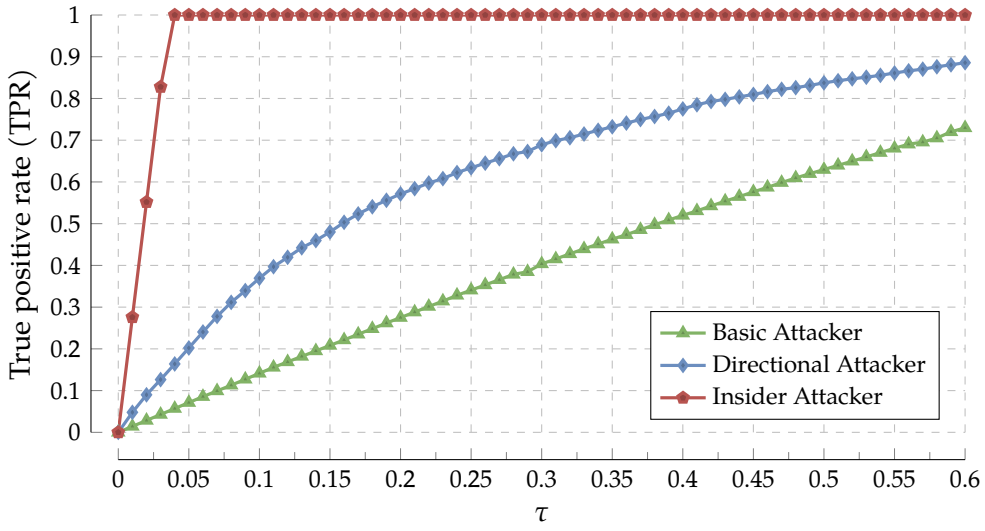


FIGURE 5.3: Results: Classification of idealized attackers/$\delta = 0$ (k-shortest paths)

$$TPR = \frac{|V_t \cap V_c|}{|V_c|} \tag{5.6}$$

$$FPR = \frac{|V_t \setminus V_c|}{|V \setminus V_c|} \tag{5.7}$$

As shown in Equation (5.7) the *false positive rate* (FPR) is calculated by dividing the number of incorrectly classified hosts by the number of host (all hosts that are not in V_c). Based on this, we can perform standard statistic analyses to determine the performance of the two variants of the algorithm and finally derive a recommended value for τ. Each experiment was repeated 50 times for both variants—one based on *k-shortest paths* and the other based on *random walks*.

5.1.5.3 Experiments

In this section, we present the experiments we conducted to evaluate our approach for reconstruction of lateral movement and discuss the obtained results.

Performance for idealized attackers In the first experiment we aim to analyze the performance of both algorithm variants for idealized attackers that follow our attacker models exactly (see: Section 5.1.2.2). The result can be treated as ground-truth to validate the core functionality of our approach.

To achieve this, we set the *deviation* δ to zero and measure the TPR for varying values of τ. Hereby, values greater than 0.6 were not evaluated for τ as it is typically not useful nor feasible to analyze more than 60% of the whole network. Additionally, this limit reduces the runtime of single experiments considerably. The results are shown in Figure 5.3 (for k-shortest paths) and Figure 5.4 (for random walks) respectively. The plots show the average TPR across 50 different graphs for each attacker class and $\tau \in [0, 0.6]$.

The reconstruction of lateral movement is most accurate for the insider attacker and least accurate for the basic attacker. The results indicate that both variants are capable

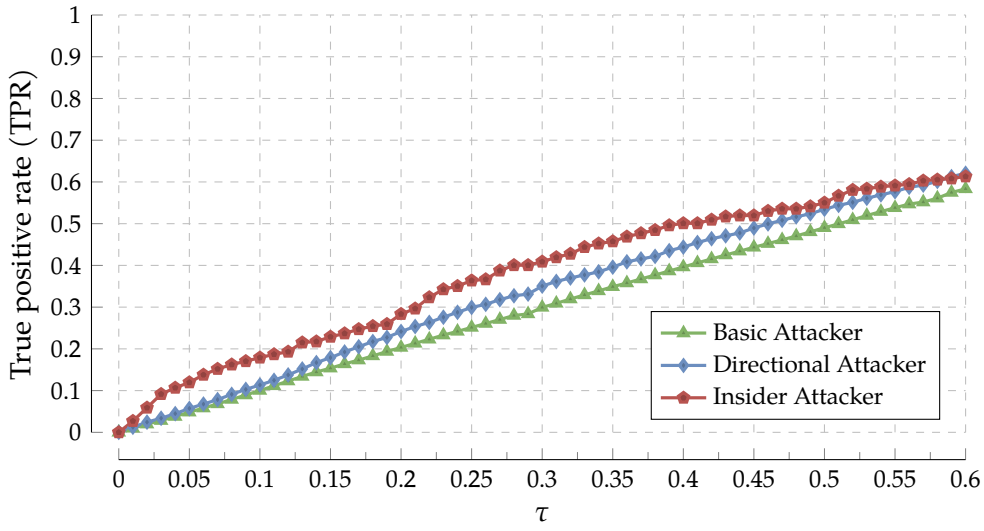


FIGURE 5.4: Results: Classification of idealized attackers/$\delta = 0$ (random walks)

of approximating the set of compromised hosts effectively with the implementation based on *k-shortest paths* performing slightly better for both basic and directional attackers and substantially better for insider attackers.

This is expected as the insider attacker strictly compromises hosts on the shortest path that is the first path returned by the k-shortest paths algorithm. While directional attacker movement cannot be fully reconstructed, the TPR is promising especially for lower values of τ. For the basic attacker the TPR is increasing slightly better than linear with τ.

The random walk based variant performs strictly inferior for all three attacker classes with almost linear performance. This is somewhat expected as δ is set to zero for this experiment, favoring k-shortest paths. However, we expected a better TPR especially for insider attackers and a higher maximum TPR for $\tau = 0.6$ and all three attacker classes.

Performance for deviating attackers Next, we aim to analyze the performance of our algorithm for more realistic attackers that deviate from the idealized models. Although the models capture expected attacker behavior, it is unlikely that a real attacker follows them exactly. Even if that were the case, we still expect some deviation, because of the different knowledge bases of attacker and security administrator (see: Section 5.1.2.1).

Therefore, in the second experiment we evaluate the influence of the deviation on the performance of both algorithm variants. The deviation parameter δ is varied in $[0,1]$ and we measure the TPR for τ set to 0.2 and 0.5 per attacker class respectively. The results are shown in Figure 5.5 (for k-shortest paths) and Figure 5.6 (for random walks) respectively. The plots show the average TPR across the 50 runs for $\delta \in [0,1]$.

The graphs indicate that δ has a small negative impact on the TPR for both intermediate and insider attackers, while basic attackers are only minimally affected. The variant based on k-shortest paths is able to reconstruct ~90% of insider lateral movement with τ set to 0.2 even when the attacker deviates 80% from the expected behavior. For both other attacker classes, the TPR basically scales linearly with the size of τ as the

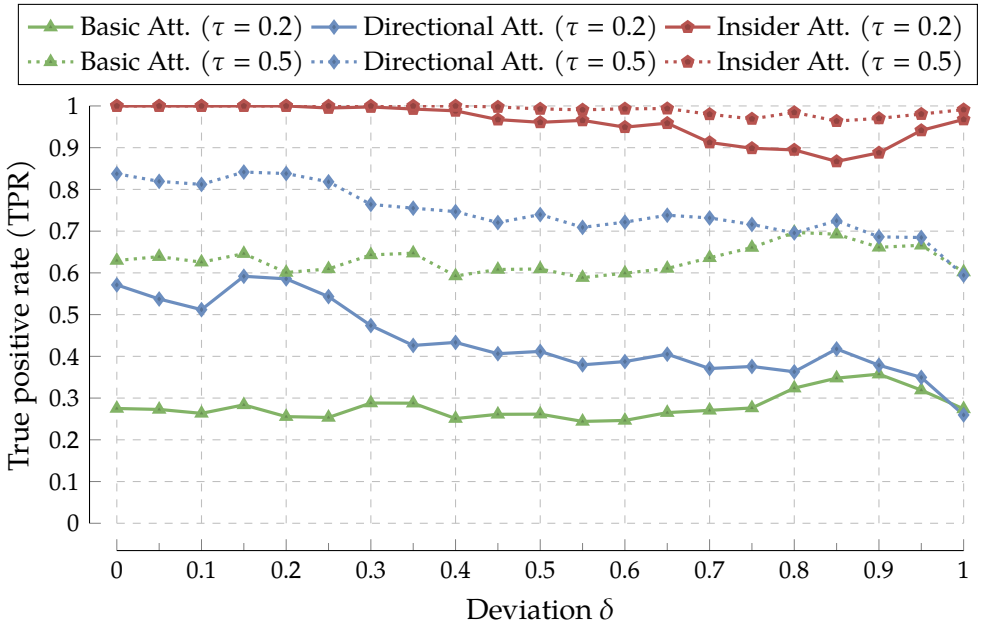


FIGURE 5.5: Results: Influence of δ on classification (k-shortest paths)

lines for 0.2 and 0.5 are basically offset by a fixed size on the y-axis. This is expected, when we look at the graphs of these attacker classes in Figure 5.3 and Figure 5.4. Additionally, the TPR is worse for the directional attacker and high values of δ while the performance for the basic attacker is nearly unaffected.

The performance of the random walk-based variant is again strictly inferior. However, it is even less affected by deviation and even sees an increase in TPR for the insider attacker and high values of δ. The influence of τ also appears to be mostly linear. This poor performance is unexpected, as we explicitly designed this variant to reconstruct lateral movement graphs that do not closely follow the attack model. One reason for this could be how the deviation parameter δ is applied to the graph. As mentioned previously in Section 5.1.5.1 δ increases the cost for a percentage of the edges to $2^{63}-1$, so that they are not chosen by the attacker. As the edges are chosen randomly, a low value for δ could result in the modification of edges that would not have been part of the lateral movement graph anyway. In these cases, the attacker compromises the same hosts as if there were no deviation. However, this mostly relates to the insider attacker, as this class naturally compromises the least amount of hosts compared to the other models (see: Table 5.1). Thus, it is least affected by our implementation of the deviation.

Recommendations for τ In the final experiment, we aim to derive useful values for τ for real-world production use. As mentioned previously τ greatly influences the runtime of the algorithm and thus should be carefully chosen. If the chosen value for τ is too low, the candidate set V_t is too small to contain all compromised hosts. High values of τ increase runtime of our algorithm and are also expected to increase the FPR. To approximate real attackers we include runs with a deviation up to 0.75 in this experiment.

We visualize the effect of τ on the performance of our approach in a receiver operating characteristic (ROC) curve in Figure 5.7 (for k-shortest paths) and Figure 5.8 (for random walks) respectively. Each data point represents a value of τ in $[0, 0.6]$.

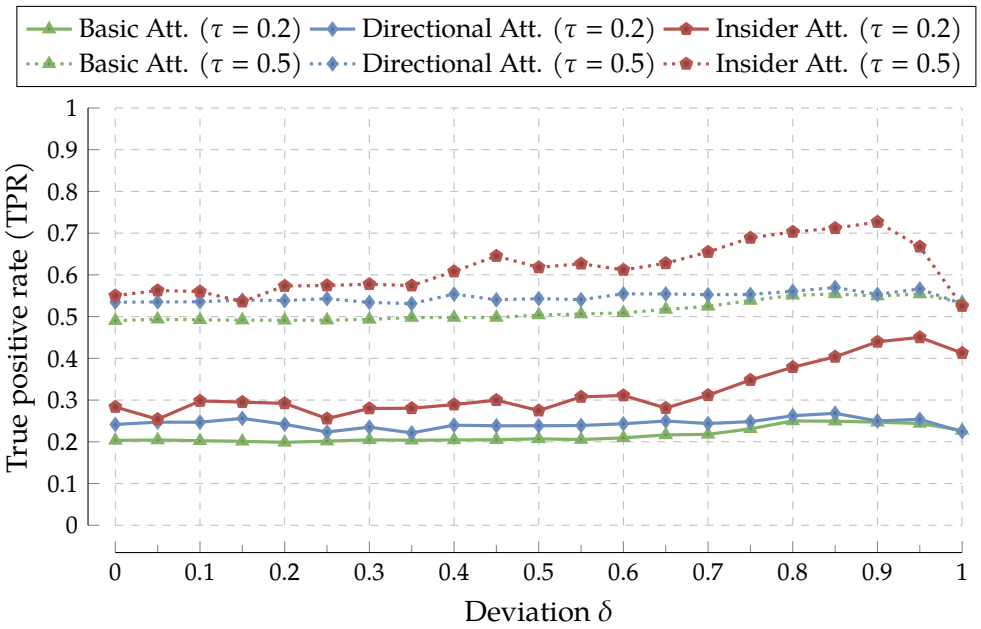


FIGURE 5.6: Results: Influence of δ on classification (random walks)

The plots also contain a graph representing a random classifier,i. e., a function that randomly classifies hosts as compromised or non-compromised. All points above this line imply a classification performance better than random. Thus, a perfect classifier would produce a result at the $(0,1)$ coordinate. Random classification is of course not a perfect approximation of real security administrators. However, it serves as a good baseline to compare both implementations.

The k-shortest paths variant achieves great performance for insider attackers with 0.88 TPR and only 0.01 FPR for $\tau = 0.05$. The performance for directional attackers is slightly less good. However, it still offers a big improvement over random classification especially for low values of τ. The graph for basic attackers also indicate a good performance trend. The ratio between TPR and FPR favors the TPR,i. e., more compromised hosts are correctly classified than wrongly classified. For low values of τ the candidate set V_c is too small to contain all compromised nodes and thus cannot offer a high TPR. The same applies for the directional attacker, although the gradient is nearly identical to the graph of the basic attacker, it decreases faster starting from $\tau = 0.15$.

The variant based on random walks offers a linear performance. For insider and directional attackers the TPR is slightly better and for basic attackers inferior to random guessing. For basic attackers this could be leveraged by inverting the classification and thus achieving a slightly better performance. However, as k-shortest paths outperformed random walks for every attacker class and deviation, we did not implement this. While these results could be expected from the previous experiment, they ultimately disqualify the random walks based variant as a useful implementation of our approach. Therefore, all recommendations that follow are given for the k-shortest paths variant.

The recommended value for τ differs depending on the amount of resources available and how many false positives should be tolerated. Also, in real-world scenarios a security administrator would not run our approach once and analyze all results but rather start from the beginning of the candidate set and rerun the algorithm once a

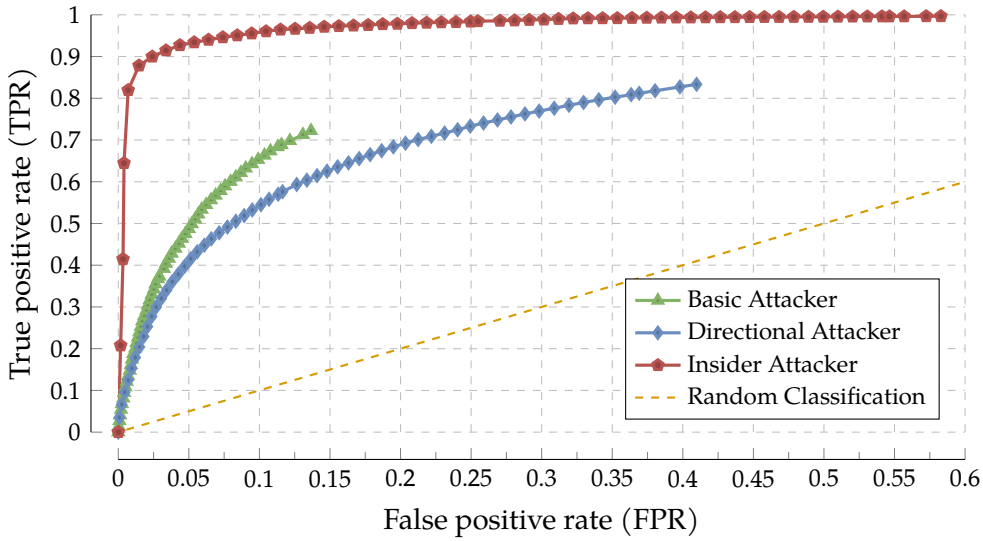


FIGURE 5.7: Results: ROC for $\tau \in [0, 0.6]$ and $\delta \in [0, 0.75]$ (k-shortest paths)

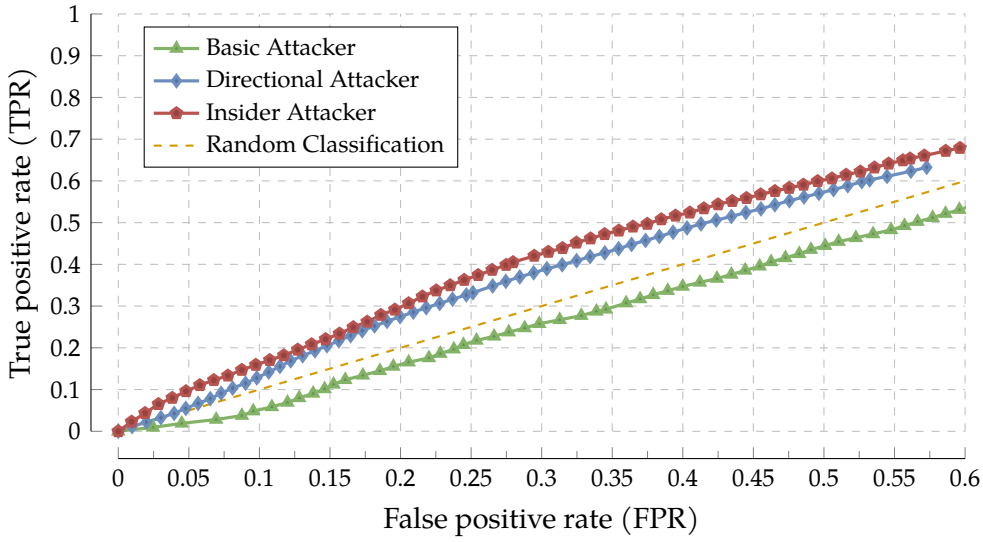


FIGURE 5.8: Results: ROC for $\tau \in [0, 0.6]$ and $\delta \in [0, 0.75]$ (random walks)

new IoC has been verified. A sensible choice for insider attackers is $\tau = 0.05$ as this achieves 0.88 TPR with only 0.01 FPR. If more false positives can be tolerated, $\tau = 0.35$ achieves 0.99 TPR with a still relatively low FPR of 0.16. For directional attacker the ideal value for τ should be between 0.1 and 0.2. Values below 0.1 contain too few results to achieve a high enough TPR while value above 0.2 start to see increased FPRs. This results in a TPR of 0.27 for $\tau = 0.1$ and 0.47 for $\tau = 0.2$. While this is certainly not enough to properly identify all compromised hosts, the resulting candidate set should contain enough compromised machines to yield more IoCs which can then be fed back into a future iteration of the algorithm.

It is generally difficult to detect all hosts that were compromised by a basic attacker, as they usually compromise many hosts in the network (see: Table 5.1). The choice of τ offers multiple points to consider. At $\tau = 0.23$ the algorithm achieves a TPR of 0.3 with only 0.02 FPR. If more false positives can be tolerated, $\tau = 0.54$ achieves 0.66 TPR at the cost of 0.1 FPR. Finally, as so many hosts are compromised anyways, it might even make sense to chose $\tau = 0.6$. This results in 0.72 TPR at still only 0.13 FPR.

In summary, our evaluation details how our approach helps to identify machines that were most likely infected during lateral movement of an APT campaign. From our two proposed implementations, the variant based on k-shortest paths consistently outperforms the one based on random walks even for highly deviating attackers, i. e., attackers that only roughly follow their assumed attacker model. Overall, performance is best for insider attackers, followed by basic attackers, and ultimately directional attackers. This is expected as the insider attacker compromised the least amount of nodes that are quickly found by our algorithm. Furthermore, we provide some recommendations for τ, i. e., the parameter that controls the size of the result set as well as the overall runtime of the algorithm. While the exact values differ between expected attacker classes, a value between 0.05 and 0.2 should be sensible for most insider and directional attackers. As basic attackers compromise large parts of the network anyway, τ needs to be significantly higher and the approach overall becomes slightly less useful. However, APT campaigns usually do not exhibit behavior similar to basic attackers and thus should remain unaffected.

5.1.5.4 Requirement Comparison

Based on the description and evaluation results, we can now compare our approach for reconstruction of lateral movement with our requirements formulated in Section 3.1. *R1: Accuracy* is marked as partially fulfilled, as the evaluation results indicate good performance for insider attackers which are conceptually similar to expected APT behavior. Due to the added context through our formal model, *R2: Explainability* is also partially met. SOC analysts can leverage the host properties (and attached metadata such as software versions) of the identifies hosts to estimate why an attacker might have attacked it. *R3: Low overhead* is fully checked as our approach only relies on abstract alerts/IoCs that mark specific hosts and thus can be very lightweight. Although not explicitly assessed in the experiments, *R4: Scalability* is partially checked as both variants offer some conceptual benefits that help to scale them. The chosen implementation of k-shortest paths efficiently caches previously found paths while random walks are completely independent from each other and thus can be massively parallelized. *R5: Security* was not explicitly measured and thus is not rated. While *R6: Privacy* was not explicitly relevant in the design of our approach, the abstract alerts do not require extensive personal information, thus partially fulfilling this requirement. Lastly, *R7: Deploy- & Maintainability* is partially met as our approach

requires some detailed information about the underlying infrastructure to establish the host properties but operates with any kind of alert/IoC afterwards, effectively simplifying deployment.

5.1.6 Summary

This section introduced an approach to reconstruct APT lateral movement activity and thus identify hosts that were likely compromised to answer our formulated research question **RQ3**:

RQ3 *How can APT mitigation be supported by combining both alerts and security-relevant host information, e. g., to estimate impacts of the attack?*

More specifically, we reconstruct lateral movement activity from an incomplete alert set based on a formal network model that captures attacker interest. The two host attributes Importance $I(v)$ and Vulnerability $V(v)$ are abstract representations of properties an attacker might be interested in. The resulting Criticality $C(v)$, i. e., the product of the first two attributes, then highlights hosts that are both vulnerable and important and thus highly attractive targets. Based on three attacker classes derived from this model, we showed an abstract algorithm to identify likely compromised hosts as well as two concrete implementations based on k-shortest paths and biased random walks. The evaluation based on synthetic attack paths revealed that the variant based on k-shortest paths consistently outperforms the random walks across all three attacker classes. Our results indicate that our approach performs best for insider attackers (that are conceptually similar to highly targeted APT attacks) and achieves about 90% TPR for attacker that deviate 80% from our idealized attacker model. Overall, the parameter τ, i. e., the size of the result set, significantly impacts both TPR and FPR. Based on our results, we recommend to set $\tau = 0.2$ for best results across all three attacker classes. Overall, our approach showed that the combination of alert data and security relevant host information can help to estimate the impact of lateral movement during an APT campaign and thus provides an answer to **RQ3**.

While our approach can help to identify lateral movement activity and thus contributes valuable alerts to the overall detection pipeline, most SOCs will perform some kind of whole-campaign detection. Several approaches in this area are based on anomaly detection and learn a model of normality. Anomalies are then classified as APT attacks and flagged accordingly. However, the underlying models are usually to complex to infer which part of the underlying data triggered the anomaly (as we discussed in: Section 3.4.2). Thus, the next section presents an approach to restore this missing context for anomaly detection-based approaches as the information is essential for efficient incident response measures.

5.2 Explainability for APT Detection on System Provenance Graphs

This section describes an approach to obtain explanations for whole-graph classification decisions made by any black box anomaly detection model through a variation of permutation importance. We apply this approach in the context of anomaly detection-based APT detection via system provenance graphs. This addresses one identified problem with the approaches discussed in Section 3.4.2, namely the missing combination of *Accuracy* and *Explainability*. Overall, this contribution aims to answer research question **RQ4** as introduced in Section 1.1:

RQ4 *Which techniques from XAI can be leveraged to add attack context to classification decisions made by anomaly detection-based APT detection approaches?*

Our approach identifies graph elements that are likely part of the detected anomaly by systematically modifying the original graph and observing the changing classification results. While we apply and evaluate our approach in the domain of graph-based APT detection, the concept should be transferable to any whole-graph classification problems based on anomaly detection, as our approach make no assumptions about the underlying model. This contribution is based on preliminary work from a Master's thesis [Wel22] supervised by the author of this thesis. Additionally, this section shares material with the corresponding publication [WWF23] that further evaluates the approach developed in the Master's thesis.

5.2.1 Motivation and Objectives

In recent years, anomaly detection approaches based on whole-system provenance data have emerged as a popular option for APT campaign detection (see: Section 3.7). The fine-grained data available in system provenance is well suited for data-hungry approaches like artificial intelligence (AI)-based anomaly detection and is able to achieve high detection performance. Systems like UNICORN [Han+20] or Pagoda [Xie+20] are able to learn sophisticated models of normality for single hosts or network segments and can detect APT activity as anomalies. After detection has happened, SOC analysts are faced with mitigation and incident response measures that require detailed knowledge about the past attack. However, the normality models that triggered the detection are usually not providing explanations beyond the detected anomaly and are too complex to be manually analyzed. Thus, analysts need to manually investigate all potentially infected systems consuming precious time the adversary can potentially remain in the network. While some approaches for XAI exist (like LIME [RSG16] or SHAP [LL17]), they have not been applied to the graph-based anomaly detectors in question as they are largely focused on tabular data as the predominant data format.

To address this problem, this contribution aims to achieve the *objective* of restoring explanations for graph-based anomaly detection approaches while satisfying the additional *requirement* of **approach independence**, e. g., by not making any assumptions about the underlying AI model. This ensures our approach is applicable to any anomaly detection approach even in other application domains.

The remainder of this section is structured as follows: Section 5.2.2 describes the overall idea of our approach based on a variation of permutation importance. Section 5.2.3 details two generic modification strategies we employ to systematically alter the original graph. Section 5.2.4 evaluates our approach based on two established system provenance datasets and compares it with the requirements established in Section 3.1.

5.2.2 Approach Overview

Our approach is based on *input permutation importance* similar to LIME. However, as our target anomaly detection approaches are based on graphs, we need to adapt the concept to systematically modify graph data instead of tabular data. While graphs can be represented as tabular data (in form of adjacency matrices), this representation is unfit to properly represent complex graphs such as data provenance graphs as the matrix grows quite large and is usually sparse which is suboptimal for modifications. Additionally, it does not include auxiliary data encoded in the graph such as node or edge properties. Figure 5.9 gives an overview about our approach. As mentioned, we

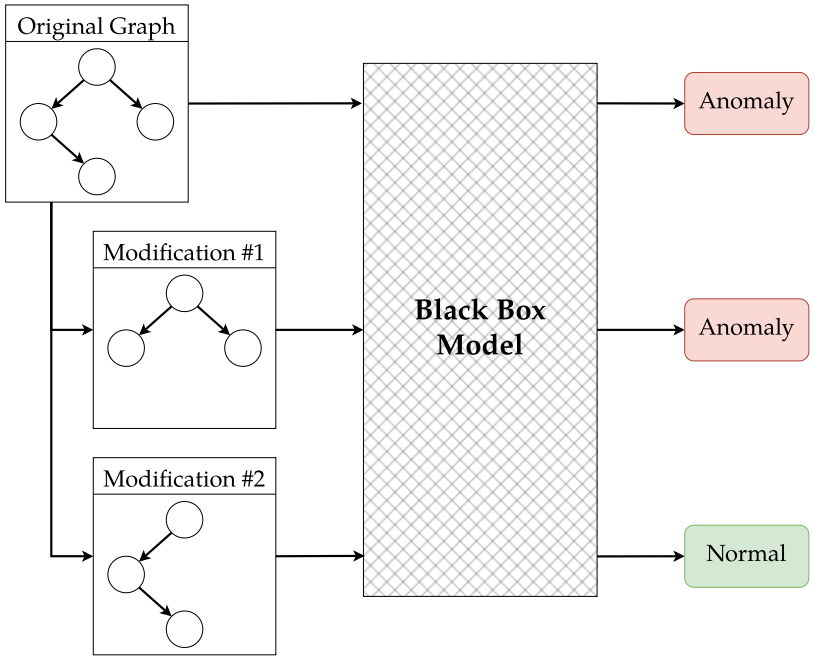
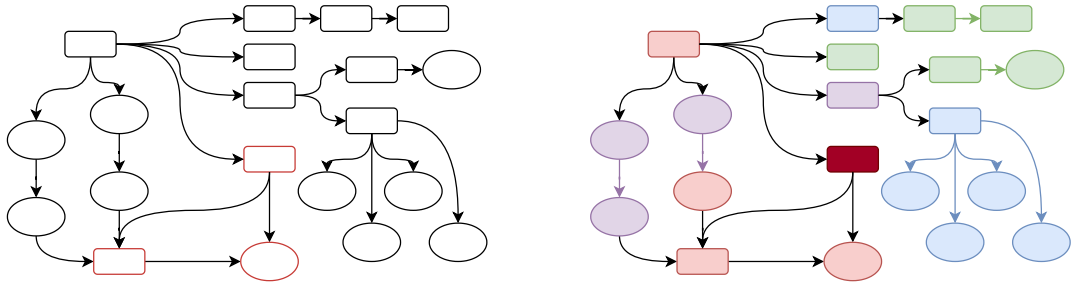


FIGURE 5.9: Overview: Explainability for graph-based anomaly detection. An original attack graph is modified several times. The modified graphs are processed with the model again. If the prediction changes from *anomaly* to *normal* the modified parts are relevant to the attack.

treat the anomaly detection model as a black box by submitting several graphs to it and observing the produced anomaly score. Once a graph is marked as anomalous, we modify it systematically and rerun the anomaly detection. If a modified graph is the detected as normal (or even just "less abnormal"), this is an indicator that the removed parts are part of the anomaly, i. e., the APT attack we aim to reveal.

Figure 5.10 visualizes the envisioned result of our approach. Figure 5.10a shows an example provenance graph that was classified as abnormal by an AI-based anomaly detector such as UNICORN. The three nodes represent attack activity and are unknown to the SOC analyst. Figure 5.10b shows the same graph after applying our approach. The different colors represent the anomaly scores obtained by removing the node from the graph ranging from green for more abnormal scores (nodes are likely benign as the score increases compared to the baseline) to dark red for least abnormal score (nodes that most likely belong to the attack we aim to reveal). This heat map-style representation helps analysts to identify relevant parts of the attack needed for further investigation, incident response, and mitigation measures.

Until now, we have not described the *modification strategies* employed to alter the original graph. There are several options such as node/edge removal or addition, node rewiring, or label changes to just name a few. However, our scenario of APT detection imposes some restrictions on our approach: (*i*) the overall result (and thus also the modification) needs to be understandable by human SOC analysts. This rules out options like node rewiring that completely change the associations between affected nodes, potentially creating invalid data provenance structures. (*ii*) the number of modifications needs to be manageable for the approach to finish in reasonable time. In our scenario of this is especially important as provenance graphs capture all system activity and thus grow quickly. This allows nearly unlimited numbers of modifications. As our goal is to identify subgraphs of the provenance



(A) Original provenance graph (B) Heatmap based on "abnormality"

FIGURE 5.10: Idea: Permutation-based explanation of subgraphs

graphs which are the result of attack activity, we focus on two concrete modifications strategies that only remove elements of the graph. The resulting anomaly scores then help to estimate if the removed elements are part of the subgraph in question. Both strategies make minimal assumptions about input graphs are therefore widely applicable while still being easy to understand.

5.2.3 Modification Strategies

This section introduces the two modification strategies we devised in the context of our approach. Our goal is to identify graph elements that are likely responsible for the underlying anomaly. Thus, we chose to focus on the systematic removal of graph elements as the changing anomaly scores then indicate if the removed elements impacted the classification decision.

5.2.3.1 Strategy 1: Node Removal

The first modification strategy operates on node-level. In our application scenario of data provenance graphs, this is intuitive as nodes in the graph represent kernel-level objects. As the attack likely is responsible for the creation of a set malicious objects (processes, files, sockets etc.), the nodes can act as "barriers" between benign and malicious system activity. Equation (5.8) formalizes this strategy. Let $G = (V, E)$ be a graph that should be explained. For each node $v \in V$ a new modified graph is created without v and its corresponding edges. The set of modified graphs is therefore given as

$$M = \{G - v | v \in V\} \tag{5.8}$$

with $-$ being the node removal operator [Die17] which simply removes one or more nodes and its edges. In short, for each node, create a new graph that does not contain the node.

This strategy has the advantage of simplicity. It is easy to comprehend, therefore identified attack patterns can be quickly analyzed and transferred into actions. Attention needs to be payed to the number of created graphs, as provenance graphs can contain large numbers of nodes. Depending on the application context, nodes may need to be prioritized to keep computation times at bay. For node prioritization there are several options which either rely on structural data alone such as node degree or on semantic data encoded in the graph such as label or other attached data.

5.2.3.2 Strategy 2: Edge Removal

The second modification strategy operates on edge-level. This resembles the streaming property of most data provenance graphs which are defined as a sequence of edges. In our application scenario, this maps closely to malicious activity such as `read` and `send` which is expressed in these edges and thus understandable to the analyst. Equation (5.9) formalizes this strategy. Let G be a graph that should be explained and is defined as $G = (V, E)$. For each edge $e \in E$ a new modified graph is created without e and any edge that follows afterwards, which is denoted by E_t. This requires edges to be timestamped or at least ordered which is the case for our data provenance graphs. The set of modified graphs is therefore given as

$$M = \{G - e - E_t | e \in E\} \tag{5.9}$$

with $-$ being the edge removal operator [Die17] which removes one or more edges.

As well as the node removal strategy introduced in the previous section, this strategy is easy to comprehend for users too. It quickly gives insights at which point during the creation of the provenance graph it became malicious. Provenance graphs can contain large numbers of edges which means that removal of every single edge is not feasible. However, as nodes are removed in an ordered fashion, they can also be removed in batches. This reduces the number of modified graphs by a factor equal to the batch size. If one batch seems to be highly relevant to the attack, the contained edges can be processed again with a smaller batch size. This can be repeated as needed to single out the relevant part of the provenance graph.

5.2.4 Evaluation

In this section, we evaluate our explainability approach for graph-based anomaly detection on two data provenance datasets that are commonly used in the domain of APT detection. First, the two datasets are introduced with some descriptive features. Next, we describe the experimental setup including the implementation based on UNICORN [Han+20] and execution environment. Finally, we present our results obtained for the explainability performance of our approach.

5.2.4.1 Datasets

For our evaluation, we chose two datasets also chosen for the evaluation of UNICORN, namely StreamSpot [MMA16] and Defense Advanced Research Projects Agency (DARPA) Transparent Computing (TC) Engagement 3 [DAR18]. Table 5.2 gives an overview about the number of attack and benign graphs as well as the average node and edge counts for both datasets.

The StreamSpot dataset was released alongside a proposed IDS [MMA16] and contains 600 graphs in total. 100 of these graphs resemble a drive-by download attack. The other 500 graphs belong to benign scenarios, e. g., downloading files, using a mail service, or playing a video game. Each provenance graph consists of system calls recorded during execution of a single scenario in an automated browser. We use the first 100 benign graphs (simulating video streaming) to train UNICORN on benign behavior.

Table 5.2: Key features of the two datasets used in the evaluation

		StreamSpot [MMA16]	DARPA TC Cadets [DAR18]
Benign	Graphs	100	109
	Mean Nodes	8,292	25,760
	Mean Edges	113,229	30,165
Attack	Graphs	100	3
	Mean Nodes	8,891	609,378
	Mean Edges	28,423	1,004,743

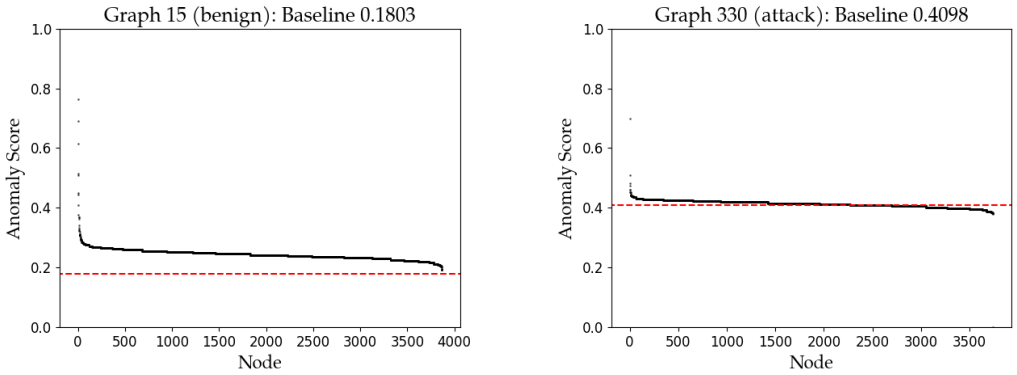
The DARPA TC Program [DAR15] aimed to provide real-world data for researchers working on intrusion detection. It provides several datasets generated by different provenance systems across five engagements. The data was gathered from real machines while security researchers executed attacks on servers and clients. This evaluation uses the data generated by the Cadets provenance tracker of engagement three [DAR18] same as UNICORN. This dataset contains three attack graphs and 109 benign graphs. It is important to note that the attacker graphs from this dataset are significantly larger than all other graphs and that there is a significant imbalance between benign graphs (109) and attack graphs (3).

Both datasets have a significant downside in the fact, that they only provide labels on graph-level. This is too coarse-grained for our evaluation that would require knowledge about which subgraph of the overall graph is responsible for the attack. StreamSpot does not provide any information in this regard besides the high-level description of the attack ("drive-by-download"). DARPA is slightly better as each engagement contains a human-readable report which describes the attacks in more detail including timestamps. However, this is not sufficient to clearly and efficiently identify the relevant subgraph. We address this problem of missing ground truth by comparing the results of our approach for benign and attack graphs via a novel metric.

5.2.4.2 Experimental Setup

As mentioned previously, we evaluate our concept on top of UNICORN [Han+20] as state-of-the-art APT detection approach based on data provenance graphs. The available implementation consists of three components that are executed in a pipeline and exchange data via certain text files. Our explainability approach was implemented as a Python wrapper around these three components and leverages the detection component as a black box function. The prototype is able to modify graphs according to both proposed modification strategies and executes UNICORN for multiple modifications in parallel for optimal resource usage.

To prioritize nodes for our *node removal strategy*, we chose the node degree property as it relies on structural graph data alone. For StreamSpot, we chose only nodes with a node degree larger than 1 as these weakly connected nodes are unlikely to be essential to an attack. In the case of DARPA TC, we increased this threshold to 5 as the relevant graphs are significantly larger. For the attack graphs in this dataset, we additionally had to employ a cutoff of 1 000 nodes to keep runtime manageable.



(A) Example benign graph (B) Example attack graph

FIGURE 5.11: Results: Examples of sorted anomaly scores obtained via node removal for two StreamSpot graphs

Early in the evaluation phase, preliminary results showed that processing of even batched modifications via the *edge removal strategy* were taking excessive time. To enable evaluation regardless, we implemented an additional variant of the modification directly on UNICORN's internal sketch data structure instead of the original graph. This shortened the runtime significantly and thus enabled the experiments. While this sacrifices our black box assumption for reduced computation time, it does not influence the explainability results as the modification on the sketches is identical to what would be produced by processing the modified graphs through UNICORN.

All experiments were run on a single bare-metal machine with 64 CPU cores and 256 gigabytes of main memory. Our prototype was instructed to use all available cores.

5.2.4.3 Results

Our approach generates an *anomaly score* for each graph modification. Depending on the modification strategy employed, the results allows us to assign an anomaly score to the removed node or edges. A lowered anomaly score indicates, that the respective graph element(s) are likely to belong to the attack (as the graph became "more normal" with the element(s) removed). Figure 5.11 shows two example plots highlighting the different anomaly scores for removed nodes. Both underlying graphs belong to the StreamSpot dataset with 15 as the example benign graph and 330 as the example attack graph. The anomaly scores were sorted descendingly and plotted on the y-axis while the x-axis shows the node index. Additionally, the dashed red line marks the *baseline anomaly score B* that was scored by the original graph and is also given in the title. We observe three key facts: (*i*) the baseline of the benign graph is lower than the attack graph as expected. (*ii*) for a large number of nodes, the removal only minimally impacts the assigned anomaly score for both the benign and attack graph (visualized by the black line closely tracking the dashed baseline in the middle of the plots). (*iii*) there are no nodes in the benign graph that decreased the anomaly score when removed, while the attack graph contains 1 476 nodes that do. This is consistent with our expectation that the benign graph only contains benign nodes (as their removal would make the model "more abnormal") while the attack graph contains a limited set of nodes caused by the embedded attack. When we consider our application scenario of APT detection, the deviation from the baseline can also be used to prioritize nodes for analysis. As our plots contain sorted data, analysts would

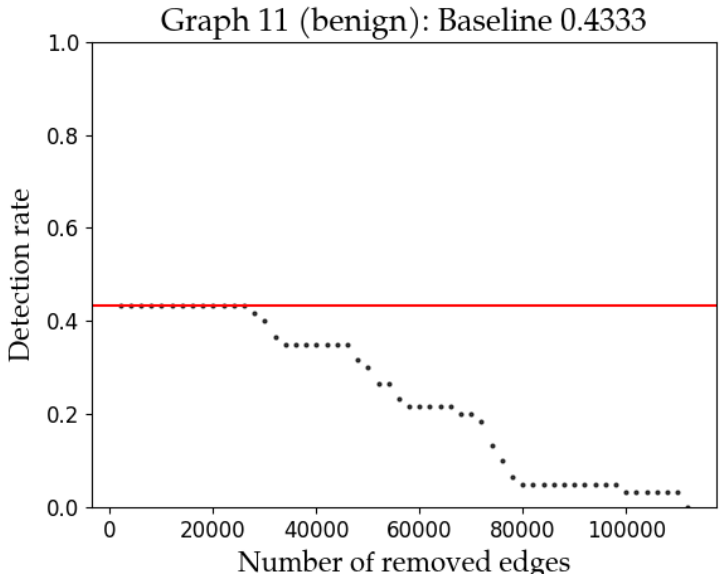


FIGURE 5.12: Results: Example of sorted anomaly scores obtained via edge removal for a StreamSpot graph

then start their investigation on the nodes farthest to the right as they produced the lowest anomaly scores.

While the anomaly scores for the node removal strategy are promising, the edge removal strategy is less convincing. Figure 5.12 shows an example plot with the edge removal strategy applied to benign graph 11 from StreamSpot [MMA16]. The x-axis again shows the detection rate, while the y-axis contains the unsorted numbers of removed edges. As mentioned previously, the edges were removed in batches of size 2000. Initially, the graph seems to indicate that many removed edge batches lower the anomaly score and thus are likely to belong to our attack. However, we also see a constant decline in the anomaly score that increases monotonically with more removed edges. This implies that the overall anomaly score generally increases with more edges added to the graph. In fact, this is true for all attack and benign graphs across both datasets: The anomaly score lowers with more removed edges starting from the baseline. As this does not offer any insightful information about the graphs but rather on the inner workings of UNICORN, we do not consider the edge removal strategy throughout the remainder of this evaluation. While the strategy might be helpful for other anomaly detection approaches or other application domains, it does not seem to produce valuable results in our scenario.

The two plots based on node removal already indicate that our approach is able to identify graph elements that are likely to belong to an attack. However, the anomaly score alone is not the best choice as a metric as it is only applicable to the respective graph element(s) and thus does not allow for comparisons between graphs. To address this, we define a new metric that describes the area below the original detection value B that we use as a baseline. We call this metric *area under baseline* (AUB) and is is calculated from the detection rates of our modifications d_i as shown in Equation (5.10). The factor of 1 000 is used for clarity and for better comparability as the original numbers are very low.

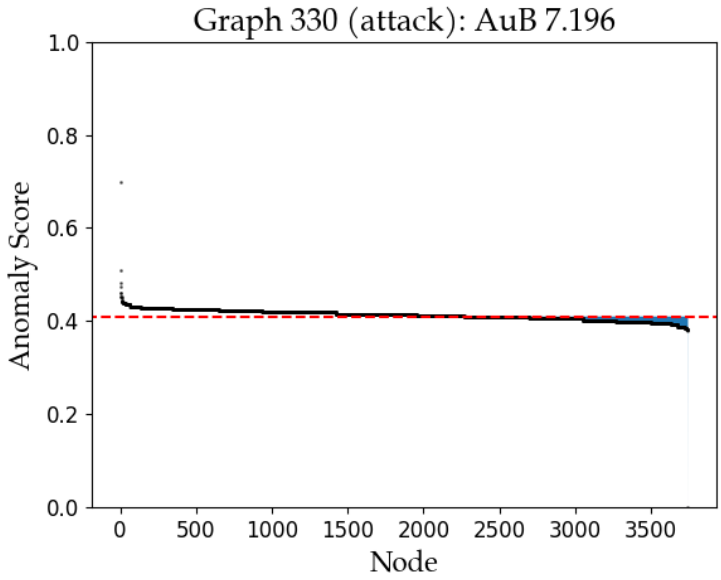


FIGURE 5.13: Results: Example attack graph with shaded area under baseline (AUB)

TABLE 5.3: Results: Median area under baseline (AUB) for node removal

Graph Type	StreamSpot [MMA16]	DARPA TC Cadets [DAR18]
Benign	0.3419	0.0000
Attack	1.0182	0.1224

$$\text{AuB}_G = -1000 * \frac{\sum_{i=0}^{n-1} max(d_i - B, 0)}{n} \tag{5.10}$$

The AUB is basically a numeric representation of the graph elements that result in lower anomaly scores when removed. The example graphs shown in Figure 5.11 in fact exactly match our expectation that benign graphs exhibit an AUB of 0 while attack graphs result in a low AUB. This is important as a high AUB would indicate that large parts of the graph are part of the anomaly and thus would need to be investigated by SOC analysts. Figure 5.13 shows the previous attack graph from the StreamSpot dataset (330) with the AUB shaded and in the title.

We can now compare the median values of the AUB across the benign and attack graphs of our datasets respectively. Table 5.3 gives an overview across both datasets and graph types. The median AUB for both datasets is larger for the attack graphs than for the benign graphs. This is expected and confirms our hypothesis that the metric accurately represents the anomalous graph elements in the attack graphs. For DARPA TC Cadets, we even see a median AUB of 0 for benign graphs, indicating that our metric represented the non-existent anomaly perfectly.

To further support our results we also analyzed the difference in AUB distribution between benign and attack graphs for statistical significance using the Wilcoxon rank-sum test [Wil92]. The test reported a p-value of 0.00004088 (0.0041%) for the results obtained from StreamSpot, strongly indicating that the difference in distribution of

AUB for benign and attack graphs respectively is indeed statistically significant. For DARPA TC Cadets, we did not conduct the test as the sample size of just three attack graphs in the dataset is too small to yield meaningful results. However, the median AUB of 0 for benign graphs is promising and indicates that the obtained values for the attack graphs are also representative of the contained anomalies.

5.2.4.4 Discussion

Our results indicate, that our approach using the node removal strategy is able to reveal graph elements that are likely to be part of the hidden attack subgraph we aim to identify. The lowered anomaly score when removing those elements suggests that the elements are part of the anomaly (i. e., attack). However, due to the missing granularity in the state-of-the-art datasets used for evaluation, we cannot completely confirm this claim. Our novel metric area under baseline (AUB) thus aims to evaluate our approach with the available labels, i. e., graph-level classification as benign or abnormal, by summarizing the identified elements. The results indicate, that attack graphs consistently score higher median values for AUB supporting our hypothesis that the summarized elements belong to the anomaly (which are not present in benign graphs). While this is not the optimal outcome, we are confident, that our approach could support security analysts in their investigations. Although some benign graphs exhibit a non-zero AUB, this is not a problem for analysts, as they would apply our approach only on graphs that the anomaly detector marked as anomalous. The resulting graph elements can then be prioritized for improved analysis and incident response.

5.2.4.5 Requirement Comparison

Based on the description and evaluation results, we can now compare our approach for explainability of graph-based anomaly detection with our requirements formulated in Section 3.1. *R1: Accuracy* is partially fulfilled as our results strongly indicate that anomalous graph elements are identified although this is not completely confirmed due to the missing fidelity of the datasets. As *R2: Explainability* was the explicit goal of our approach, it is fully met. The added contexts supports SOC analysts in identifying relevant details about the detected anomaly. *R3: Low overhead* is partially checked as our approach does not require additional data but incurs some significant computations. As runtime performance was not explicitly measured, *R4: Scalability* is not rated. Similarly, *R5: Security* is not directly applicable to our approach and was thus not evaluated. *R6: Privacy* remains an open question in any approaches based on system provenance graphs. However, our approach does not impact this requirement directly thus marking this requirement as neutral. Lastly, *R7: Deploy- & Maintainability* also remain unsolved as the reliable capture and processing of high-volume system provenance data still poses significant operational challenges. As our approach again does not impact this requirement, it is also rated neutral.

5.2.5 Summary

This section presented an approach to obtain explainability for graph-based anomaly detection approaches through a variation of permutation importance to answer our formulated research question **RQ4**:

RQ4 *Which techniques from XAI can be leveraged to add attack context to classification decisions made by anomaly detection-based APT detection approaches?*

The added context about specific classification decisions enables SOC analysts to better understand the detected attack and informs incident response measures. Additionally, our concept is also applicable to whole-graph classification problems in other application domains as we treat the anomaly detection function is treated as a black box and no assumptions about the underlying model are made. More specifically, we systematically modify the graphs that were classified as anomalies and observe the changing anomaly scores. If the result is "more normal", we likely removed (part of) the anomaly. Our evaluation leverages UNICORN [Han+20] as the anomaly detection black box and is based on two established datasets in the system provenance domain: StreamSpot [MMA16] and DARPA TC [DAR18]. While we cannot fully confirm our explainability results due to the lack of fidelity in the datasets, we devised a metric to quantify the anomalous graph elements and evaluate our approach based on that. Our analysis shows statistically significant deviations in median values between benign and attack graphs (as confirmed using the Wilcoxon rank-sum test [Wil92]) as expected. Overall, our obtained results indicate that our approach successfully leverages a variation of permutation importance to identify likely anomalous graph elements and thus provides an answer to **RQ4**.

The presented approach helps to obtain context about detected attacks and significantly supports SOC analysts. However, one issue with system provenance-based APT detection approaches remains in the form of difficult deploy- and maintainability as the reliable capture and processing of this kind of high volume data is challenging. The next section addresses this problem by suggesting an alternative approach for whole-campaign APT detection that requires only minimal information in the input alert set. Additionally, it can ingest both regular untagged alerts as well as alerts with associated stage label as produced by stage-specific approach as discussed in Section 3.6 or presented in the first section of this chapter.

5.3 APT Contextualization via Kill Chain State Machines

This section introduces a novel approach to contextualize APT activity according to the UKC [Pol21]. The algorithm is based on a Kill Chain State Machine (KCSM) that describes multi-stage attack campaign progress in its states and contains attack stages from the UKC as transitions between them. The resulting scenario graphs are compact representations of potential APT activity that support SOC analysts in their investigations. Overall, this contribution aims to answer research questions **RQ5** and **RQ6** as introduced in Section 1.1:

RQ5 *How can established kill chain-based APT models be leveraged for campaign detection with only minimal assumptions about the lower-level alert set?*

RQ6 *Which level of volume reduction can be achieved during APT campaign reconstruction to lessen the impact of alert fatigue on SOC analysts?*

Our approach ingests network-based alerts (both with and without attached APT stage labels) and links them according to the pre- and post-conditions of the state machines. After several optimization steps, we obtain compact APT scenario graphs that offer extended context about the reconstructed campaign activity. Additionally, we describe an extended version of our approach that includes prioritization measures that further support SOC analysts in working with the obtained scenarios.

This contribution is based on preliminary work from a Master's thesis [Ort19] supervised by the author of this thesis. Furthermore, this section shares material with

the corresponding publication [Wil+21] that significantly revises and extends the approach developed in the thesis. The optimizations and approaches for scenario prioritization described in Section 5.3.5 were developed in a Bachelor's thesis [Lau22] that was supervised by the author of this thesis.

5.3.1 Motivation and Objectives

Campaign-based APT detection is a promising area of research that aims to detect the presence of APT adversaries by analyzing their activity as a whole. This is achieved by either detecting *anomalies* that diverge from normal system activity as APT activity or by *reconstructing potential campaigns* based on an underlying APT model. However, both approaches from both areas have significant shortcomings (as described in Section 3.7): (*i*) anomaly detection approaches fail to explain which malicious activity triggered the anomaly, thus effectively hindering incident response measures, (*ii*) current reconstruction approaches are based on system provenance data that is expensive to capture and difficult to reliably maintain, (*iii*) most reconstruction approaches use outdated models that are too limited to accurately describe current APT activity, and (*iv*) most approaches from both categories are not extensible and thus hard to integrate into existing detection environments.

To address this problem, this contribution aims to achieve the following *objective*: detect APT activity on campaign level by reconstructing the campaign according to a comprehensive model. In contrast to prior approaches from literature, the following *requirements* should be satisfied (in addition to the overall requirements formulated in Section 3.1):

- **Self-contained stage mapping**: The approach needs to be able to deduce potential APT stages from the input data without relying on any external detection systems. This ensure that the approach can be deployed in greenfield scenarios without any existing security infrastructure.
- **Extensibility**: In addition to *1.*, the approach should nonetheless integrate with potentially existing detection approaches and integrate their generated IoCs & alerts into the reconstruction process. This is especially helpful for stage-based detection approaches that may offer higher detection accuracy for a single APT stage compared to the provided stage mapping.
- **Lightweight data sources**: To simplify deployment in large organizations, the approach should not be based on system provenance data. Alternative lightweight data sources like network traffic or coarse-grained host data is easier to reliably capture and should thus be preferred.
- **Compact representation**: The representation of the reconstruction is aimed at SOC analysts that need to further investigate if the detected scenario is (part of) an actual APT campaign is present in the network. To support the analyst as good as possible, the representation should be compact enough to quickly gain a good overview about the potential campaign while maintaining references to all underlying data for further analysis.

This section describes our approach for multi-stage attack detection. We first apply alert correlation to preprocess alerts into clustered meta-alerts and unclustered single alerts. Next, we feed both types of alerts to our APT detection and contextualization approach. We assign potential attack stages to alerts and link subsequent stages according to our *KCSM* to generate *APT scenario graphs*. These scenarios reveal potential

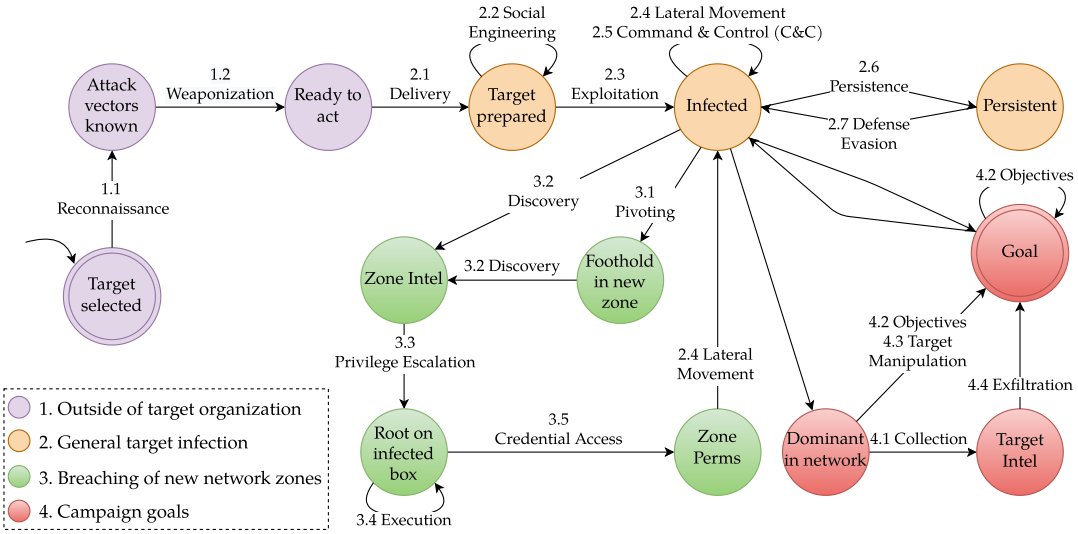


FIGURE 5.14: Kill Chain State Machine (KCSM) derived from UKC [Pol21]

multi-stage attacks present in the alert data and aid the human analyst during incident triage and mitigation.

The remainder of this section is structured as follows: Section 5.3.2 introduces the KCSM that we derived from UKC [Pol17; Pol21]. This formalization makes the existing model more actionable and lays the groundwork for our approach. Section 5.3.4 describes our algorithm to characterize and detect APT and other multi-stage attacks. We show how potential attack stages can be derived from single alerts and meta-alerts and leverage the state machine to connect these attack stages to expressive *APT scenario graphs*. Next, Section 5.3.5 describes some extensions to the base approach that improve the algorithm and help to prioritize the generated APT scenarios. In Section 5.3.6 we evaluate both the base approach and the extended prototype on synthetic and real-world data and discusses the obtained results in the context of the requirements established in Section 3.1.

5.3.2 The Kill Chain State Machine (KCSM)

A segregation of a network into multiple network zones, which is standard in security-sensitive organizations, forces an APT actor to follow a kill chain model [BYG14]. The UKC [Pol17; Pol21] is the most comprehensive model from related work and was thus chosen as basis for our approach. The original UKC involves 18 stages (see: [Pol21]) that might manifest in different host and network activity. Some stages might occur repeatedly, others might be left out. Nonetheless, there is a certain ordering of attack stages that cannot be changed. For example, malware *Delivery* necessarily comes before *command & control* (*C2*/*C&C*). Likewise, some stages might reoccur such as *lateral movement* (*LM*). Additionally, *Delivery* can happen via the network in the form of an email or offline via an infected thumb drive. *Exploitation* is a host-level activity and *Pivoting* is often visible on the network and hosts likewise. Monitoring every individual device of an organization is almost impossible [Mar+16], but monitoring the entire network of an organization is feasible even though expensive. As about half of all UKC APT stages and transitions between them are visible on network level, we limit our approach to these stages, enabling us to work on network data only.

To make the model actionable for our algorithms, we formalize the stages and their ordering to a finite-state machine in Figure 5.14 that we call the *KCSM*. We map APT stages to transitions as they represent attacker actions. States in KCSM represent the campaign progress with "*Target selected*" as the common start state. As the goals of APT campaigns can be quite diverse, the end state is abstractly labeled "*Goal*". Overall, both states and transitions are divided into four major categories:

Outside of target organization (purple) An APT campaign starts when the attacker has *selected the target*. *1.1 Reconnaissance* follows to determine possible attack vectors and to choose an attack strategy. The attack is tailored to the target and malware is *1.2 Weaponized* with an exploitable bug.

General target infection (orange) Once the APT actors are ready to act, they *2.1 Deliver* the weapon to the target, e. g., malicious Dropbox implant or a water-holed web server. The next goal is to get the delivered weapon executed in the target network. Therefore, tactics like *2.2 Social Engineering* might be employed. After the initial *2.3 Exploitation*, KCSM reaches the central state of infection. From here on, all further APT actions take place inside the target's infrastructure. The central state of "Infected" allows an APT to take various paths. Colors in KCSM states indicate different action paths that can be pursued. The states for infection and persistence are marked orange. Whenever a new box is infected, malware may be *2.6 Persisted* and *2.7 Defense Evasion* techniques are applied. It is possible to take actions in form of *2.5 C2/C&C* and *2.4 lateral movement (LM)*, just circling in the infected state. LM is considered the movement to a known target. Identifying new targets, however, falls into the third category.

Breaching of new network zones (green) An APT might need to *3.1 Pivot* from an infected box or might already be in a position where it is worthy to start *3.2 Discovery*. The *3.2 Discovery* action is similar to *1.1 Reconnaissance* but is conducted in the internal networks of an organization. Thereafter, the zone is known. Once the attacker *3.3 Escalates Privileges*, they may *3.5 Access Credentials* to gain network-wide permissions. After that *2.4 LM* is possible within the new network zone, which brings the attacker back to the general state of infection. *Note*: The KCSM makes the entire process of network zone breaching an optional path. Controlling the infection or spreading it to new machines, is always possible via taking the circular *2.4 LM* and *2.5 C&C* transitions in the "Infected" state. However, this path would presume that the attacker possess comprehensive knowledge of the network.

Campaign goals (red) The last category of states marks the path for acting towards objectives. From the state of infection, the attacker can reach a predominant network position or even directly proceed to the campaign goal. These unlabeled transitions indicate that APT campaign goals can be highly diverse. It is not required for an APT to *4.1 Collect* or *4.3 Manipulate* data. Similarly, *4.4 Exfiltration* is not required to happen. However, in case data is exfiltrated, the attacker is required to collect the targeted data first. The unlabeled backwards transition from "Goal" to "Infected" state symbolizes how APTs remain in a system as long as possible. Acting on objectives, exploring new network zones, controlling and spreading are not necessarily bound to ever finish.

It is also important to note that we can differentiate between stages that *compromise new hosts* and others that do not. The simplest example for the first category would be

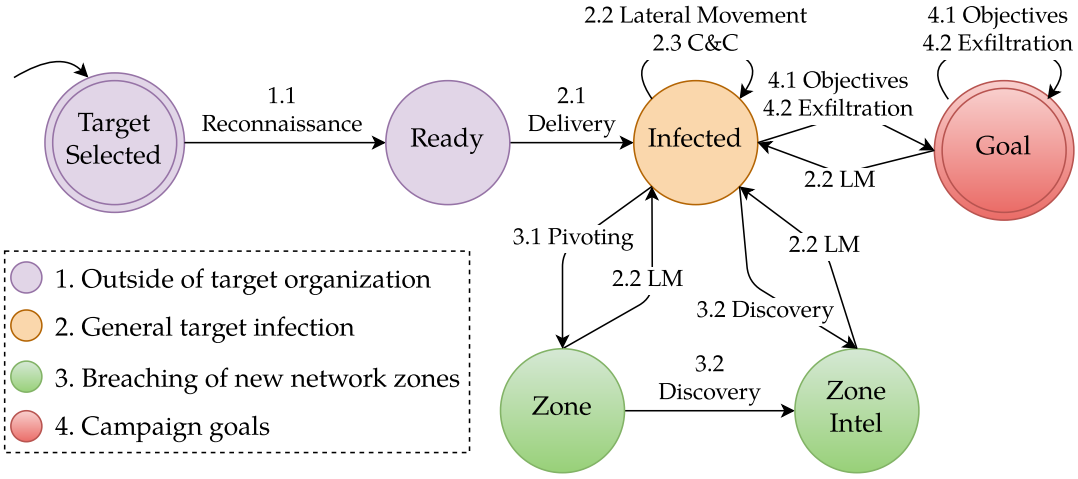


FIGURE 5.15: Network Kill Chain State Machine (NKCSM) derived from UKC [Pol21]

2.1 Delivery while *2.3 Command and Control* does not compromise a new host. We will use this information later in the detection algorithm approach to improve the results.

APT stages can be observed in many parts of a compromised network. *2.1 Delivery* can happen via the network in the form of an email or offline via an infected thumb drive. *2.3 Exploitation* or *3.3 Privilege Escalation* are solely host-level activities and *3.1 Pivoting* is often both a network and a host action. Monitoring the entire network of an organization is expensive. But monitoring every individual device of an organization is almost impossible [Mar+16]. With a network monitoring system (NMS) like *Zeek* [Zee22] it is possible to monitor the network traffic even of large networks with little administrative overhead. In contrast, collecting, shipping, and unifying traces from thousands of heterogeneous system components is more challenging. Interestingly, about half of the state transitions (and thus APT stages) might be observable on network level. The other half of all transitions may be observed on host level or even outside of the organizations scope. To highlight this, we present the *Network Kill Chain State Machine* (*NKCSM*) in Figure 5.15, a version of the KCSM reduced to all stages that might be detectable at the network level.

The NKCSM keeps the start and end states from the original state machine (*Target Selected* and *Goal*). APT stages that either occur outside of the target's network (*1.2 Weaponization*, *2.1 Social Engineering*) or on host level (*2.3 Exploitation*, *3.3 Privilege Escalation*, *3.4 Execution*, *3.5 Credential Access*, *2.6 Persistence*, *2.7 Defense Evasion*, *4.1 Collection* and *4.3 Target Manipulation*) are excluded. This reduces the total number of states from thirteen to six, while preserving core semantics of KCSM. The numbering of APT stages is preserved for simplicity. The general progression from initial infection over optional lateral movement and zone breaching to action on objectives is still present in NKCSM. However, all remaining transitions are network-based and thus potentially observable via a NMS.

5.3.3 Approach Overview and Example Scenario

Figure 5.16 gives an overview about our approach to contextualize APT campaigns. The input consists of both *tagged alerts* (alerts that are labeled with a set of potential APT stages it represents) and *untagged alerts* (regular alerts that are either host- or network-based) as well as a set of network *zone definitions* that describe the segmented zones of the network. In the first phase, the untagged alerts are preprocessed in

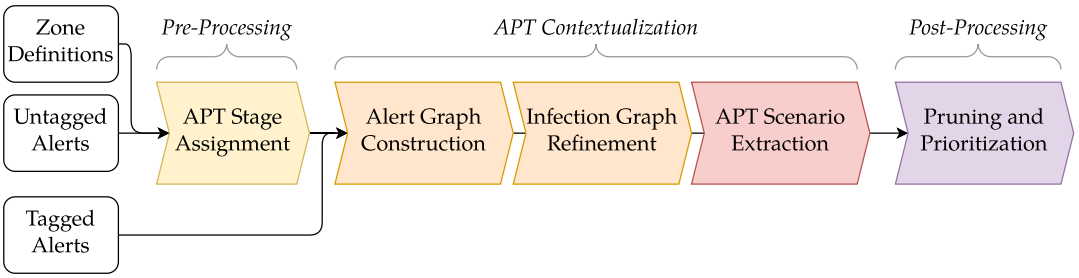


FIGURE 5.16: Overview: APT contextualization via KCSM

the APT stage assignment and received stage labels based on their innate network direction. Next, all alerts contain stage labels and can be linked to *construct alert graphs* based on the pre- and post-conditions of the KCSM. The resulting graphs are then *refined to infection graphs* through aggregation and elimination of superfluous information. In the last phase of the APT contextualization, the approach *extracts APT scenario graphs* from the set of infection graphs. Finally, the resulting set of scenarios is optionally post-processed by *pruning and prioritization*. The resulting set of APT scenarios is then ready for manual investigation by SOC analysts. We now describe each of these phases in more detail along a small example APT campaign.

For demonstration purposes, we present a small artificial multi-stage attack scenario, that we use as an example throughout this section to illustrate the reconstruction process. In this scenario, the target network is divided in two zones Z_1 (10.1.0.0/16) and Z_2 (10.2.0.0/16). The attacker aim to find valuable hosts in Z_2 and uses three attacking machines with public IP addresses (4.4.4.4, 1.3.3.7, 1.4.4.7) during the campaign. While a real APT campaign would perform more attack steps, especially after key hosts are discovered, this example is intentionally kept small. The attacker performs the following five steps in order:

1. *Reconnaissance:* The attacker uses an Internet host to scan four hosts in Z_1 (4.4.4.4 → [10.1.0.1, 10.1.0.2, 10.1.0.3, 10.1.0.4]).
2. *Delivery:* The attacker uses another Internet host to deliver a malware-dropper to a host in Z_1 (1.3.3.7 → 10.1.0.4).
3. *Delivery (Download):* The infected machine downloads a second-stage malware from the Internet (10.1.0.4 → 1.4.4.7).
4. *Pivot:* The malware pivots to two hosts in Z_2 (10.1.0.4 → [10.2.0.1, 10.2.0.3]).
5. *Discovery:* The malware scans the three remaining machines in Z_2 for valuable services (10.2.0.3 → [10.2.0.2, 10.2.0.4, 10.2.0.5])

5.3.4 APT Contextualization

This section presents the algorithm for APT campaign reconstruction based on the KCSM as described in the conference publication [Wil+21] (that was based on the supervised Master's thesis [Ort19]) up to basic scenario pruning that is applied to the final result set. We implemented this approach as a Python-based prototype, that we refer to as **APT Contextualizer v1**.

Full name	Abbreviation	Network direction
Reconnaissance	**R**	$Z_0 \rightarrow Z_i$
Delivery	**D1**	$Z_0 \rightarrow Z_i$
Delivery (2nd stage DL)	**D2**	$Z_i \rightarrow Z_0$
C2/C&C	**C**	$Z_i \rightarrow Z_0$
Lateral Movement	**L**	$Z_i \rightarrow Z_j$
Discovery ("Scan")	**S**	$Z_i \rightarrow Z_j$
Pivoting	**P**	$Z_i \rightarrow Z_j, \ i \neq j$
Exfiltration	**E**	$Z_i \rightarrow Z_0$
Objectives	**O**	$Z_i \rightarrow Z_j$

Table 5.4: Network-visible stages of APT attacks

5.3.4.1 APT Stage Assignment

The states and transitions formalized in the KCSM can be used to trace a multi-stage attack campaign. However, we still lack a way to map network alerts to potential APT stages. Looking back at the KCSM, we realize that all APT stages still present in the state machine are characterized by movement between specific network zones. Thus, we can use the *network direction* of an alert to derive potential attack stages it can represent. We define the network direction of a (meta-)alert to be the transition between the network zones of the source and destination IP of the alert. A network zones describes one or more subnets of the same trust level. Examples of typical network zones in an enterprise network are `external`, `intranet`, and `dmz`. Given the topology information of the target network, we can assign the network direction for any network-based alert. This makes our approach extremely flexible as it does not require special stage information in our source alerts.

Table 5.4 shows which network directions are found in each stage of APT attack stage. As expected, there is an overlap between the stages, e. g., a meta-alert from Z_0 (Internet) to Z_i might be an indicator for either Reconnaissance or Delivery. Thus, our goal is not to assign a single APT stage to each alert and meta-alert but rather a set of potential stages. Given the information from Table 5.4, we obtain four distinct sets of potential APT stages depending on the network direction of the alert. Outgoing connections to the Internet (Z_0) are labeled with [D2, C, E] and incoming connections with [R, D1]. Internal connections are tagged either with [L, S, O] if both hosts are part of the same zone or [L, S, P, O] if the zones differ between source and destination host.

5.3.4.2 Alert Graph Construction

With the implied transition order from KCSM and the mapping of alerts to potential APT stages, we can build a directed graph based on pre- and post-conditions of each state. Each meta-alert results in a node with the potential stages, sources and targets of the attack attached as additional labels. Two nodes u, v are connected with an edge e if the following three conditions are met. The resulting edge e is then labeled with the set of APT stage preconditions,i. e., the set of APT stages of u that represent a precondition to any stage of v.

1. The *latest* timestamp of u is smaller than the *earliest* timestamp of v.

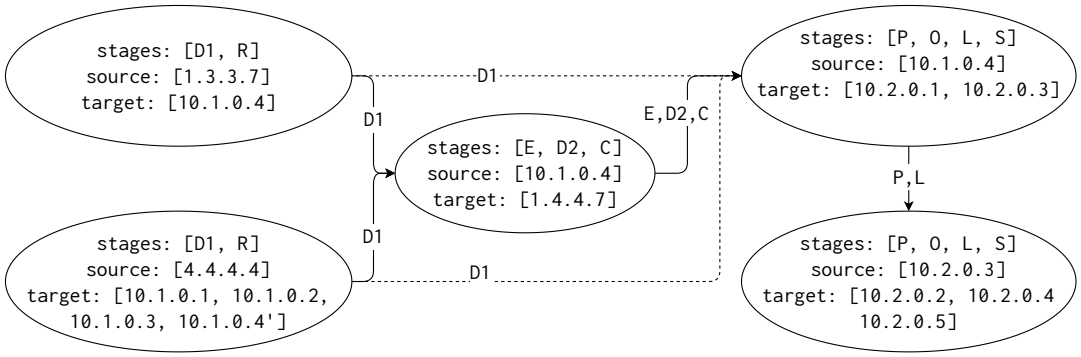


FIGURE 5.17: *Alert graph* comprising five alerts generated from the example scenario

2. The potential APT stages of u contain at least one stage, that is a precondition for any APT stage of v. If multiple stages match, the ones that *compromise new hosts* are preferred.
3. The *source* and *target* IP addresses of u and v match according to the APT stages of u. For stages that *move infection* (D1, L, P) the *target* address of u matches a *source* address of v. Other APT stages, that do not compromise new hosts, are valid if any *source* IP address overlaps between u and v.

Figure 5.17 shows the alert graph that contains five alerts derived from the stages in our example. For simplicity we assume that an IDS produced a alert for each malicious action without any false positives. The alert graph already shows promising results. The path the attackers took throughout the zones is clearly visible. However, we can already see at least two suboptimal properties of the graph: First, the APT stage labels on the edges are broad and thus sometimes contain more APT stages than only the correct one. This is not a problem, as the more relevant information, i. e., IP addresses and alert identifiers, is contained in the nodes anyway. Second, the alert graph contains edges that do not add any useful information (indicated by dashed lines in the figure) as they lie on a path shorter than the longest path between two nodes. This means, they discard the additional information that is present on the nodes that are not included on these paths. While it is possible, that this represents the actual attack and the skipped node is the result of a false positive alert, the longest path offers more information to the threat hunter (namely the hosts to investigate for further indicators of compromise). Additionally, this type of graph can grow quite large quite easily. Due to the loose requirements for linking two nodes, resulting graphs can be almost fully connected in real-world scenarios. This would not efficiently support human threat-hunters as they cannot extract meaningful information anymore. However, our example graph is relatively compact, it only represents a small attack without any false positives.

5.3.4.3 Refinement to APT Infection Graphs

We can *reduce* the density of the alert graph without losing important context by *aggregating* existing information and *eliminating* obsolete information. We start the graph consolidation on a node that does not possess any outgoing edges. We can guarantee the existence of at least one such node due to our timestamp requirement on the edges (condition 1 for connecting two nodes in the alert graph). Starting from this node, we recursively iterate through the incoming edges while aggregating those with identical APT stage labels. Longest paths are preferred during the iteration and

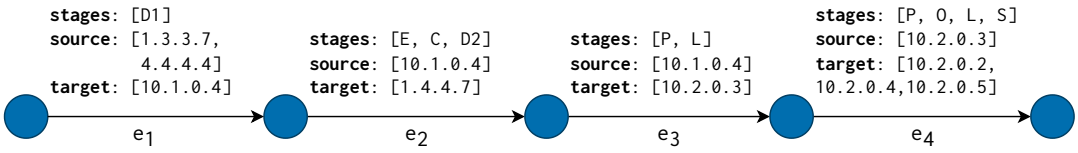


FIGURE 5.18: *APT infection graph* for the example scenario without false positives

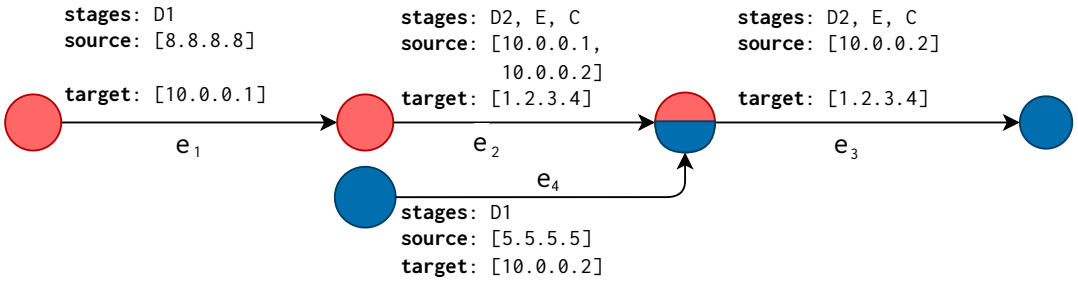


FIGURE 5.19: Example for a transitively invalid *APT infection graph*

paths shorter than the longest path between two nodes are discarded. The *source* and *target* sets of the connected nodes are combined for matching edges to obtain the set union.

The compact graph obtained by this process is called *APT infection graph*. In this graph, nodes mark APT campaign progress, while the edges represent the APT stages with the related information such as involved IP addresses and alert identifier. Figure 5.18 shows the APT infection graph for our example scenario. It is significantly smaller than before, while retaining the important information about IP addresses and stages. The progress of the potential APT campaign is clearly visible, and the reduced branching helps focusing on the important information for threat hunters. The combination of the set of potential APT stages and IP addresses directly hints at how the hosts should be further investigated as the different stages usually leave distinct indicators of compromise on the machine. When we described KCSM, we differentiated between stages that compromise new hosts and stages that do not. Until now we only used this information to prioritize stages while constructing alert graphs. As a result, the APT infection graph might contain edge pairs that represent consecutive stages in KCSM (and thus are valid in the infection graph) but have a non-infected host as the source for any following stage. An example for that is given in Figure 5.19.

In the figure we see that host 10.0.0.1 gets infected from the Internet (e_1). This is followed by outbound connections from the hosts 10.0.0.1 and 10.0.0.2 (e_2) and another outbound connection from 10.0.0.1 only (e_3). While the stage sequence D1 → [D2, E, C] → [D2, E, C] is valid, the host 10.0.0.2 was never infected on the red path (e_1, e_2) and thus cannot be responsible for the outbound connection in e_3. If we follow the blue path (e_4, e_3) the host is infected, but the source IP for the preceding *Delivery stage* differs (5.5.5.5). Thus, the graph actually contains two distinct potential APT scenarios that partially overlap in hosts that should be extracted separately.

5.3.4.4 APT Scenario Graph Extraction and Basic Pruning

Overall, *APT infection graphs* provide a decent overview about potential multi-stage attack campaigns. However, we saw that they can still contain invalid edges compared

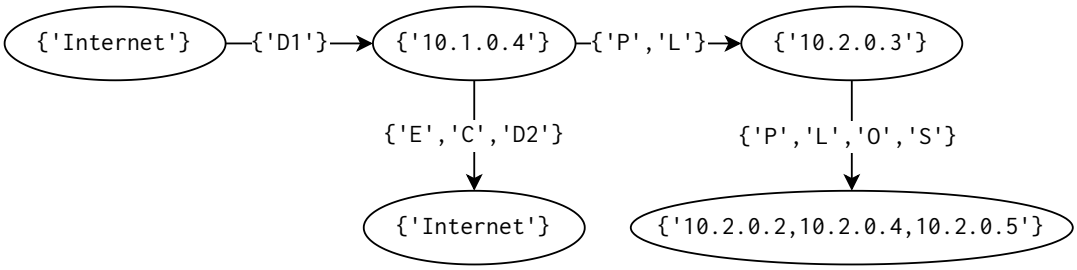


FIGURE 5.20: *APT scenario graph* representing the example APT campaign

with our original state machine KCSM or even multiple APT scenarios as separate paths. To address this, we introduce another optimization step to obtain the final graph representation for potential APT scenarios.

APT scenario graphs can be obtained by extracting transitively valid paths from APT infection graphs. The structure closely resembles the KCSM. Edges are labeled with sets of potential APT stages while nodes contain the involved IP addresses in the respective attack steps. Figure 5.20 shows the APT Scenario graph that the algorithm produces for our example scenario. The graph is concise and supports the human analyst in combating the *alert fatigue* by highlighting the affected internal hosts and the progress of the potential APT campaign. External IP addresses are replaced with the *'Internet'* label to limit the potential state expansion. These are of secondary interest to the analyst anyway, as they can only directly investigate internal hosts. However, they are not "lost" as the full alert data is available through the alert identifiers attached to the graph (left out here for brevity). To sum up, the APT scenario graph not only represents a *significant reduction* from the original alert set it was generated of but also offers essential context about the campaign progress to human analysts.

The final set of *APT scenario graphs* can be further reduced and optimized. Due to the process for graph construction, we can sometimes obtain two graphs, where one is an *isomorphic subgraph* of the other. In our context, this means, that both graphs describe the same potential APT scenario and the larger graph just contains additional steps that are not present in the smaller one. Similar to the edge elimination step from *alert graphs* to *APT infection graph*, we can eliminate these smaller graphs entirely without losing relevant information. While the problem of subgraph isomorphism is known to be NP-complete, it can often be solved efficiently. Our prototype uses the implementation from the Python library `networkx` that is based on Cordella's work [Cor+04] to prune and deduplicate the final result set. Additionally, some very basic semantic pruning is performed to further eliminate scenarios that are unlikely to be of interest to the analysts such as "very short scenarios" that only contain two nodes. While our first prototype does not perform any sophisticated optimization, a real-world deployment can add additional post-processing of the final set of APT scenario graphs, e. g., prioritization of scenarios that contain a specific critical host or scenarios with the longest chains. As APT and other multi-stage attacks are highly dynamic and tailored to the target, we want to emphasize the opportunities for further optimization here, but leave the actual implementation to the respective target organization.

Te recap, we implemented the approach described so far as a prototype in Python that we refer to as **APT Contextualizer v1**. It is based on the reduced NKCSM, can derive potential APT stages for any network-based alert and produces APT scenario graphs that are checked for isomorphic subgraphs and run through the simple pruning process described in the previous paragraph. This prototype was evaluated as part of

the corresponding publication [Wil+21] which we describe further in Section 5.3.6.2. The following section describes extensions to this base prototype that improve both overall performance as well as improve prioritization of the result set.

5.3.5 Extensions and Scenario Prioritization

The approach presented in the previous section is able to generate helpful APT scenario graphs that can be used by SOC analysts to quickly obtain an overview about potential APT campaigns in the network. However, the result set remains rather large and analysts have to be able to efficiently prioritize which graph to investigate first. This section presents extensions to the base approach as well as scenario prioritization measures aimed at reducing the load on SOC analysts. These approaches were developed as part of a supervised Bachelor's thesis [Lau22] that extended and applied the contextualization prototype in a real-world enterprise environment. The resulting implementation is referred to as **APT Contextualizer v2**.

5.3.5.1 Extensions

The extensions presented in this section are based on learnings from applying the prototype in a real-world enterprise setting and aim to improve the overall detection performance for such environments. The first extension adds support for two host-based APT stages to the implementation while the second improves the approach's ability to consume low-level event data without significant performance degradation.

General Improvements The **APT Contextualizer v1** represents a fully-functional implementation of the presented contextualization approach. However, as typical with academic implementations, it is strongly tied to the original evaluation environment and thus not easily usable in other contexts. **APT Contextualizer v2** improves on that by modernizing the code and introducing Python interfaces for several key steps in the algorithm such as data ingestion, optimizations, and result prioritization. The original code was adapted by the student that wrote the corresponding Bachelor's thesis and the latest version is available on Github [WOL22].

Integration of basic host-based stages The enterprise environment that was used to evaluate the approach, features two key host-based alert types: a general *host* alert and a more specific *objectives* alert. The full KCSM, as presented in Section 5.3.2, covers the entire UKC [Pol21] and includes all its 18 stages. Thus, the overall contextualization approach conceptually supports all stages also. However, the **APT Contextualizer v1** described in the previous section focuses on network-visible stages by using the reduced NKCSM. While the implementation could be extended to match the complete state machine this would mean that most if not all host-based stages would need to be detected to prevent "incomplete scenarios" that end in a certain part of the state machine as no alert for the following stage is generated. The alternative is to slightly adapt the currently used NKCSM to include the two new stages as additional information. This resulted in the Extended Network Kill Chain State Machine (ENKCSM) as show in Figure 5.21.

Compared to the NKCSM, the ENKCSM features three major changes: (*i*) The delivery stage is split into two phases with the second one being optional, thus resulting in two "Infected" nodes. This resembles a change that was already present in the **APT Contextualizer v1** implementation but not yet reflected in the model. (*ii*) The "host"

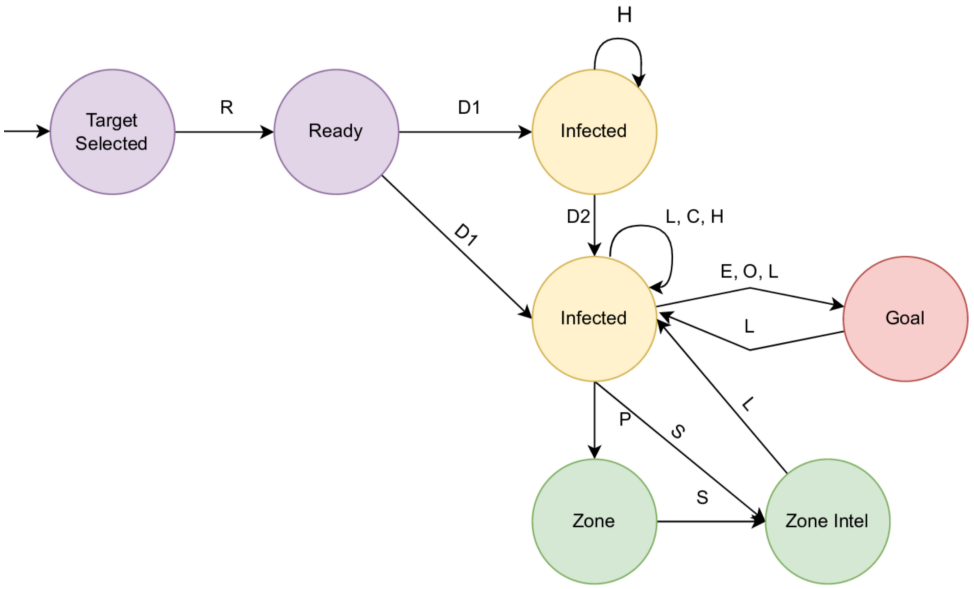


FIGURE 5.21: Extended Network Kill Chain State Machine (ENKCSM) [Lau22]

stage is added as reflexive edges on both "Infected" nodes. This enables the approach to optionally attach one or more host alerts to the chain and thus enrich it without impacting the contextualization process of the following stages. (*iii*) The "Objective" stage is conceptually extended to also cover host alerts. While this does not change the state machine itself, it is important to mention as the original NKCSM was explicitly limited to network-based objective actions.

Graph connectivity-based approach for event enrichment The enterprise environment used to evaluate the approach features alerts for several network- and host-based alerts. However, certain stages such as *lateral movement* are not currently covered by regular alerts. The environment does however provide event-level logs for some benign events that could be the result of such missing stages, e. g., legitimate-looking SSH or RDP connections. While the original approach was designed to potentially ingest such events, e. g., as events tagged with *lateral movement* as the matching APT stage, this naive implementation proved to be unfeasible in the high-volume enterprise environment—the runtime of the contextualization process increased massively to the point of being unusable. To address this, an alternative method for such event enrichment was devised based on the concept of *graph connectivity*.

The basic idea is to (*i*) find two nodes across all infection graphs that could be potentially connected (based on the pre- and post-conditions of the kill chain) and (*ii*) find a potentially matching event that "fills this gap" based on timestamps. If a matching event is found, it is used to connect the two nodes (and the rest of their respective infection graphs) and saved as a separate new infection graph, i. e., all three infection graphs (the two original ones containing S and D as well as the connected one) are kept. This is important as the event may be a false-positive thus potentially creating an invalid scenario.

$$(C_{pre}(S) \cap C_{post}(D)) \wedge (ts(S) < ts(D)) \tag{5.11}$$

$$ts(S) < ts(E) < ts(D) \tag{5.12}$$

More formally, we aim to connect a source node S with a destination node D via an event E based on the pre- and post-conditions C and timestamp ts. Equation (5.11) shows the first condition that is first checked to find potential node pairs to match. If at least one node pair is found, the second condition shown in Equation (5.12) is checked for all available events to find matching transitions. All matched events $E \in E_{match}$ are then connected in new infection graphs in the form of $S \rightarrow E \rightarrow D$ and used throughout the remainder of the contextualization process.

With the graph connectivity approach, **APT Contextualizer v2** is able to efficiently leverage event-level information to connect partial infection graphs with stages that did not produce an alert. As the event lookup is only performed selectively if a potentially missing edge is found, the implementation can retain the runtime performance required for real-world scenarios.

5.3.5.2 Scenario Prioritization

The extensions described in the previous section already improve the quality of the final set of APT scenario graphs. Adding host-based alerts and leveraging events produces graphs that are able to describe complex campaigns more closely. However, the problem of efficient scenario prioritization still remains as SOC analysts are starved for time and need to use their available resources effectively. A prioritization strategy helps to rank APT scenario graphs based on their estimated threat level. This section describes an approach for scenario prioritization via an aggregated prioritization score composed of four subscores with configurable weights:

1. **Asset Risk**: This subscore is based on the notion of notion of distance to asset (DtA), i. e., the number of required network transitions to compromise a key asset in the network. The DtA is based on the concept of reachability between network zones and the assets contained within. In contrast to the transitions in the state machines, that resemble general campaign progress across zones, the zone transitions used to calculate the DtA are actual network zones that need to be crossed to reach a target zone and asset, i. e., 0 if the attacker has already compromised the asset, 1 if the attacker has access to a zone that in turn has direct access to the asset in question, and $n + 1$ if the attacker is n zones away from accessing the asset. The risk subscore for one APT scenario graph is then calculated via (i) detailed information about zone access rules (to calculate the number of required transitions), (ii) the critical assets (as predefined information from the configuration file), and (iii) the compromised hosts in the scenario. This subscore thus represents the estimated threat of the APT campaign to the business goals of the organization.

2. **Attack Authenticity**: This subscore aims to approximate the likeliness of the APT scenario graph in question being an actual attack based on known graph patterns that are commonly encountered. For each pattern found in the scenario, the subscore increases. The current implementation of **APT Contextualizer v2** supports three "universal" patterns that are mostly independent from the environment it runs in, but can be extended with additional environment-specific ones if so desired. Currently supported are these patterns: (i) *known infection start*, i. e., graphs that contain a start node with a valid start stage, (ii) *pivoting and scanning*, i. e., graphs that contain at least one instance of these optional stages as this indicates higher attacker sophistication, and (iii) *host alerts after zone transitions*, i. e., graphs that contain a host alert after a zone transition, as the

compromise of a new host should trigger a host-based alert (assuming perfect visibility).

3. **Number of Infected Hosts**: This subscore is simply based on the number of infected hosts as a rough indicator of potential damage the campaign might cause. Although this might not be true for stealthy APT attacks, which might inflict high damage with minimal infections, a large number of hosts usually results in increased mitigation efforts as machines need to be reinstalled or otherwise cleaned.
4. **Scenario Length**: This subscore is another simple metric counting the number of nodes in the graph as an indicator of scenario complexity. APT scenario graphs are an aggregated summary of the underlying alert data they were generated from. As same-stage activity is grouped, longer graphs are the result of an increased number of distinct stages across the whole campaign. This subscore therefore provides a rough indication of the complexity and "completeness" of the campaign.

Each of the four scores is calculated and weighted to obtain the overall prioritization score. While the final score is normalized to [0,100], the subscores are not required to fit to this range. This is an explicit design choice to enable individual subscores to drastically influence the overall prioritization in severe cases. Furthermore, the system offers an "escape hatch" to assign the highest priority for two special cases: (*i*) compromise of a key asset (identified by IP address) or (*ii*) presence of key stages, that are deemed especially critical in the target environment. This ensures that APT scenario graphs that are likely to massively damage the organization can be prioritized independent of their other properties.

5.3.5.3 Summary

In summary, **APT Contextualizer v2** improves upon the initial prototype **APT Contextualizer v1** implemented as part of [Wil+21] by (*i*) increasing overall code quality and modularity, (*ii*) adapting the NKCSM to the ENKCSM to include two host-based stages, and (*iii*) providing a configurable mechanism to prioritize APT scenario graphs through an extensible prioritization score. The resulting implementation is thus better suited for real-world deployments and will likely produce improved results in any scenario.

5.3.6 Evaluation

We evaluate our approach for APT contextualization in two main aspects: *volume reduction* and *campaign reconstruction*. The first experiment demonstrates how our approach can reduce overall alert volume and thus lessen the burden on SOC analysts. To calculate this, we divide the number of obtained *APT scenario graphs* by the total number of alerts produced. Both the reduction and number of scenarios (in total and per day) are key metrics to estimate the impact on day-to-day SOC operations. The second experiments evaluates the functional performance of campaign reconstruction. For this synthetic APT campaigns are injected into the respective dataset and processed by the approach. The final set of APT scenario graphs (that has potentially been filtered) should then contain the campaign.

We evaluate both the original approach (**APT Contextualizer v1**) as presented in Section 5.3.4/[Wil+21] on the CSE-CIC-IDS2018 dataset [SHG18] as well as the extended

Type	Network traffic (pcap)
Timespan	14.02.–02.03.2018 (10 days; some skipped)
Hosts/Subnets	450/6
Connections	64 151 422

Table 5.5: Dataset Overview: CSE-CIC-IDS2018 [SHG18]

version (**APT Contextualizer v2**) described in Section 5.3.5/[Lau22] on the real-world enterprise datasets obtained in the associated Bachelor's thesis. The remainder of this section is structured as follows: Section 5.3.6.1 introduces the two datasets and matching synthetic APT campaigns. In Section 5.3.6.2 we evaluate the original approach via CSE-CIC-IDS2018 for both volume reduction and campaign reconstruction. Section 5.3.6.3 does the same for the extended version based on the real-world enterprise datasets and Section 5.3.6.4 discusses the obtained results in the context of the requirements established in Section 3.1.

5.3.6.1 Datasets and APT Scenarios

This section introduces both datasets used throughout our evaluation and their respective synthetic APT campaigns that were carefully designed to closely resemble potential real-world attacks.

Datasets We designed our approach to cover the whole detection pipeline from security monitoring to security analytics, i. e., to ingest raw network traffic and produce high-level APT scenario graphs. To the best of our knowledge, there are no public network traffic datasets that contain real APT campaigns. In combination with the highly dynamic nature of APT campaigns this renders the evaluation of detection and campaign reconstruction approaches difficult. We address this problem by using a well-known dataset for intrusion detection, namely CSE-CIC-IDS2018 [SHG18], and injecting a custom APT campaign into it.

The CSE-CIC-IDS2018 dataset is intended for the evaluation of network IDSs and contains several unrelated attacks as well as benign traffic. Table 5.5 shows some key characteristics of the dataset. The scope of six internal zones as well as 450 hosts matches a small to mid-sized company. The duration of ten days is quite small when considering APT scenarios, however it still allows for a multi-step attack campaign with certain delays in between. Although no APT campaign is present in the original dataset, some of the seven contained attacks resemble single steps of an APT campaign, e. g., an infiltration attack. Additionally, some other attacks, such as distributed denial of service (DDoS) or password guessing via brute-force usually produce large amounts of traffic and a large number of alerts unrelated to the APT campaign. This makes it a great candidate for our evaluation as both types of unrelated alert should be filtered by the reconstruction process as no connecting stages can be found.

The second dataset used for evaluation of the extended prototype was captured on alert-level during the supervised Bachelor's thesis that also developed the extensions [Lau22]. As it is based on real traffic from customers of a mid-sized managed-security provider, the data was only available on alert-level and pseudonymized to avoid leakage of potential sensitive information. Table 5.6 shows some key characteristics about the two variants of the dataset "Enterprise-Small" and "Enterprise-Large".

	Enterprise-Small	Enterprise-Large
Type	— Alerts (network- and host-based)—	
Timespan	25.07.–25.08.2022 (30 days)	26.05.–25.08.2022 (90 days)
Servers	— ~13 000 —	
Clients	— ~130 000 —	
Alerts	17 726	54 220

Table 5.6: Dataset Overview: Enterprise Datasets [Lau22]

The datasets span 30–90 days with the small variant being entirely contained in the large variant. The captured network segments contained about 13 000 servers and 130 000 client machines. The captured network traffic and host logs are processed by an unspecified security information and event management (SIEM) system that generates both network- and host-based alerts. The small dataset consists of 17 726 alerts while the large variant contains 54 220.

Synthetic APT Scenarios To evaluate the detection performance of our contextualization approach, we designed a realistic APT campaign. As the source dataset contains network traffic over ten days, we could not exceed this duration in our scenarios. We named this campaign **IDS2018-APT** and its attack steps are summarized in Table 5.7.

The APT campaign proceeds as follows: On day 1 the attackers perform a remote code execution (RCE) via the *EternalRomance* exploit on a vulnerable host in the R&D department of the target organization with the IP address 172.31.64.67. Thereafter, the malware downloads a second stage trojan from a compromised Internet host with the IP address 12.34.12.34 and persists on the machine. Three days later, the malware performs C2/C&C communication to the attacker-controlled Internet host 1.1.14.47. On day 5 the malware moves laterally to a server machine 172.31.69.20 via `PsExec`, a tool for legitimate Windows remote administration. On day 10 the compromised server exfiltrates a copy of a core database to the attacker-controlled Internet host 1.1.15.57. Overall, IDS2018-APT involves four malicious Internet hosts (1.1.13.37, 12.34.12.34, 1.1.14.47, and 1.1.15.57) and two internal hosts that are targeted by attacks (172.31.64.67, 172.31.69.28). The movement across two zones resembles a potential APT campaign in which the attackers had inside knowledge about the network segmentation and directly moves to the target host.

The APT campaign was injected into the CSE-CIC-IDS2018 dataset, which contains benign traffic as well as some unrelated attacks. The pcap files for the attack steps

Day / Date	Attack	Source	Target
01 / 14.02.18	RCE via *EternalRomance*	1.1.13.37	172.31.64.67
01 / 14.02.18	2nd stage trojan download	172.31.64.67	12.34.12.34
04 / 20.02.18	Cosmic Duke C2/C&C	172.31.64.67	1.1.14.47
08 / 28.02.18	PsExec via SMB	172.31.64.67	172.31.69.20
10 / 02.03.18	Data exfiltration via HTTPS	172.31.69.20	1.1.15.57

Table 5.7: Campaign Overview: IDS2018-APT

Date	Attack	Source	Target
26.07.22	Delivery via an zero-day exploit	2.2.2.2	10.61.99.69
11.08.22	C2 Communication via HTTPS	10.61.99.69	2.2.2.1
16.08.22	SSH Connection with stolen credentials	10.61.99.69	10.59.22.3
16.08.22	Process hollowing to prepare deployment	— 10.59.22.3 —	
20.08.22	Domain controller access via *kerberoasting*	10.59.22.3	10.61.4.3
25.08.22	Domain account creation	— 10.61.4.3 —	

Table 5.8: Campaign Overview: Enterprise APT

were collected from various web sources: *ericconrad.com* [Con17] for the *EternalRomance* RCE and trojan download, University of Twente [Uni17; Bor+17] for the *Data Exfiltration Malware* samples and more specifically *Cosmic Duke* C2/C&C traffic and *github.com/401trg* [40117] for pcaps containing *PsExec via SMB*. The exfiltration was performed locally and the traffic recorded via *tcpdump* [Tcp22]. As mentioned before all attacks were chosen on the basis of previous exploitation in the wild, e. g., PsExec is commonly used for lateral movement in typical Windows-based enterprise networks at the time of writing. The pcap files were then rewritten in both IP addresses and timestamps via *tcprewrite* [KA22] and Wireshark's *editcap* [Wir22] to match our APT campaign steps. This process resulted in a coherent attack scenario that took place at the time of the original CSE-CIC-IDS2018 dataset (February 2018–March 2018).

For our enterprise dataset we also designed a synthetic APT campaign, this time on alert level. The concrete attack steps were designed by the student based off his experience as part of a SOC from a managed security provider. All internal IP addresses have been anonymized as was performed on the whole enterprise dataset.

The APT campaign proceeds as follows: On the first day of the campaign, the adversary delivers initial malware from an infected Internet host (2.2.2.2) via a zero-day exploit (a recent example would be *Log4Shell* aka CVE-2021-4422[1]) to an internal host (10.61.99.69). Next, the malware idles for sixteen days before proceeding to perform C2/C&C communication to another Internet host (2.2.2.1). The timespan is deliberately chosen to avoid naive detection measures that only consider data from the last fourteen days. Another five days later, the attack spreads via SSH to another internal host in another subnet (10.59.22.3) by using stolen credentials. Immediately after, the adversary uses process hollowing to disguise their steps and prepare for the following deployment. Four days later, the attacker accesses the domain controller via the *kerberoasting* attack. Finally, after another five days have passed, the attack concludes by creating an account on the domain controller effectively gaining access to the managed zone.

Overall, the enterprise APT campaign involves two malicious Internet hosts (2.2.2.2 and 2.2.2.1) and three internal hosts that are targeted by attacks (10.61.99.69, 10.59.22.3, and 10.61.4.3). The targeting of a domain controller represents a typical goal when targeting enterprise networks as it provides access to the whole network zone it manages.

Construction of the enterprise APT campaign was done on alert-level as the underlying production network, the alerts were generated in, was not accessible. Thus, one alerts was generated for each campaign step resulting in six total alerts. Naturally, this

[1] see: https://www.cve.org/CVERecord?id=CVE-2021-44228

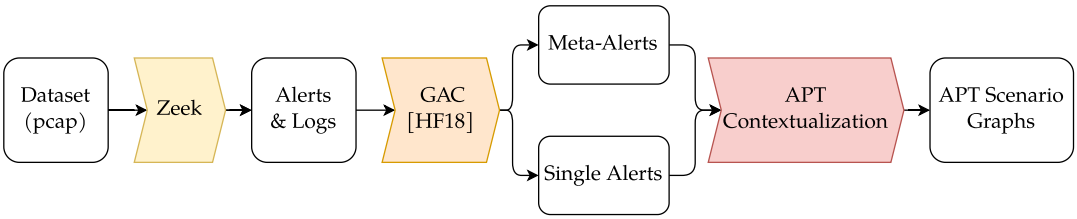


Figure 5.22: Experimental Setup: CSE-CIC-IDS2018

assumes that all activity is actually observed by a sensor. Furthermore, all evaluation on the enterprise dataset (with and without the presented APT campaign) was done on-premise by the student and not accessible to us for compliance reasons.

5.3.6.2 CSE-CIC-IDS2018

This section contains the evaluation of our original approach for APT contextualization via KCSMs (**APT Contextualizer v1**) based on the CSE-CIC-IDS2018 [SHG18] and our embedded synthetic APT campaign IDS2018-APT.

Experimental Setup We prepared an evaluation pipeline starting from scenario pcap files and resulting in a set of generated APT scenario graphs. Figure 5.22 given an overview about the involved components and artifact. The pcaps were processed by *Zeek* as network-based IDS, resulting in alerts and log files. Next, we used Graph-based Alert Correlation (GAC) [HF18] as an established alert correlation algorithm. This resembles a real-world deployment with Zeek as IDS and some alert correlation to reduce alert volume. The processing with GAC yields *clustered meta-alerts* and *unclustered alerts*. Both sets of alerts are persisted in *Elasticsearch* [Ela22] for later use in the APT contextualization. Our prototype is written in Python and leverages *networkx* [Net+22] for graph processing.

For alert generation we use Zeek v3.1.4 in two distinct configurations: MIN and FULL. MIN loads almost all scripts that Zeek offers per default. Exceptions include file extraction, as the transferred files are not needed in the process, and SSL/TLS/OCSP validation, as these scripts produce false positives for the self-signed certificates in CSE-CIC-IDS2018. In addition to the default scripts, we wrote some Zeek scripts tailored to our organization scenario including detection of downloaded Windows executables and large outgoing connections. These organization-specific scripts are *not* written to exactly detect the attack steps of our APT campaign directly, but rather match the overall scenario of a mostly Windows-based enterprise network. Thus, they also produce false positives especially as the traffic unrelated to our APT campaign also consists of the same protocols that the scripts monitor, e. g., SMB. MIN therefore resembles an organization that customizes Zeek to some extent, however, does not include any third-party scripts for improved visibility.

The FULL configuration loads all scripts from MIN and adds two well-known third-party scripts, namely *Bro/Zeek ATT&CK-based Analytics and Reporting* (*BZAR*) [The22a] for the detection of adversarial activity related to the Mitre ATT&CK framework and *0xl3x1/zeek-EternalSafety* [0xl22] to detect potentially malicious SMB protocol violations that are used in the *Eternal** family of Windows exploits. FULL therefore resembles a realistic setup in high-security environments, in which security administrators perform threat hunting to detect APT campaigns. As mentioned in Section 2.2.3 the Mitre ATT&CK framework is well known in the community and aimed at detecting

Level	Metric	**MIN**	**FULL**
Zeek	Alerts	12 675	446 407
GAC	Meta-Alerts	119	10 713
	Singleton Alerts	4 432	39 649
	Alerts for Contextualization	4 551	50 362
APT	APT infection graphs	442	456
	Total APT scenario graphs	3 452	4 305
	Distinct APT scenario graphs	611	686
	Volume reduction	4.82%	0.15%

Table 5.9: Results: APT contextualization for IDS2018-APT

attack steps of complex multi-step attacks such as APT campaigns. Both configurations are examples for organizations with different threat profiles. MIN includes less scripts and might therefore miss important attack steps. FULL includes scripts that will result in an increased alert volume with more false positives, which will also complicate successful detection of APT campaigns. The configurations highlight the range of scenarios our approach addresses.

We chose GAC [HF18] as state-of-the-art approach for alert correlation to cluster alerts. However, any correlation algorithm can be used as both meta-alerts and single alerts are are used as input to the following APT contextualization process. GAC was run in batches on all alerts of a day in the dataset. It is important to note, that our approach is not strongly tied to any alert correlation or APT stage detection scheme. Other alert correlation approaches can be used and stage-specific alerts for any APT stage are supported.

Experiment I: Volume Reduction In the first experiment, we aim to evaluate how the contextualization process reduces the data volume SOC analysts need to manually investigate. To measure this, we run our approach on the unmodified CSE-CIC-IDS2018 [SHG18] dataset and compare the number of generated APT scenario graphs with the size of the original alert set that was used as input. *Note*: As the unmodified dataset does not contain any APT attack traces, all results can be considered false positives. However, the volume reduction is especially important in this setting, as the dataset contains data from a typical mid-sized enterprise environment and the number of generated scenarios helps to estimate the amount of time that needs to be invested by SOC analysts.

Table 5.9 summarizes the obtained results the of the contextualization process for both Zeek configurations on the unmodified dataset. For the MIN configuration 12 675were generated by Zeek resulting in 611deduplicated APT scenario graphs—a significant reduction to 4.82%. For the FULL configuration, the reduction percentage is further improved to 0.15(based on 446 407alerts and 686scenario graphs). This result especially highlights how the IP-based aggregation of alerts throughout the process helps to lower the amount of generated scenarios. Considering our ten day timespan from the dataset, the number of scenarios would require the SOC to investigate 61–69 scenarios per day, which seems feasible given the dataset contains 450 hosts across 6 subnets and **APT Contextualizer v1** only performs basic deduplication and pruning.

Experiment II: Campaign Reconstruction The first experiment already provided some promising results related to the level of volume reduction that our approach is able to achieve. In this section, we aim to evaluate the reconstruction performance of our prototype based on IDS2018-APT, the synthetic APT campaign we embedded into CSE-CIC-IDS2018 [SHG18]. To fully understand how the different stages of our evaluation pipeline (as shown in Figure 5.22) impact the final result, we discuss each stage starting from the alert-level that was generated by Zeek up to the final set of APT scenario graphs.

Table 5.10 shows all alerts generated by Zeek for the two different configurations grouped by their *alert type*. The first block contains alerts generated by scripts that were shipped with Zeek. The second block contains all scripts written by us tailored to the organization scenario (prefixed with "Org"). The third and fourth blocks are only included in the FULL configuration and contain alerts produced by *MITRE's BZAR* (prefixed with "ATTACK") and *EternalSafety* (prefixed with that name) respectively. For each alert type and configuration, the table lists lists the total number of alerts observed of that type, the total number of alerts that were produced as a result of our injected APT campaign, and the ratio between all alerts of that type and the ones related to the campaign. This ratio is the TPR related to our APT campaign. The remaining alerts are the result of either attacks unrelated to the APT (as present in the original dataset) or false positives. The table shows that the overall TPR is very small with 0.0237% for MIN and 0.000245% for FULL, respectively. The type of alerts provides some insight about which parts of our APT campaign were detected. The organization-specific script related to executable downloads over a non-HTTP channel revealed the second stage trojan download on day 1, while the SMB script detected the PsExec lateral movement on day 8. The alert `Org::Very_Large_Outgoing_Tx` was generated for the exfiltration on the last day. Overall four out of the 12 675 total alerts produced by the MIN configuration relate to IDS2018-APT in some way. In the FULL configuration, *BZAR* adds seven additional APT related alerts that are all caused by the lateral movement. *EternalSafety* produces three more APT related alerts as a result of the initial *EternalRomance* RCE on day 1. While the alerts `EternalSafety::DoublePulsar` and `EternalSafety::EternalSynergy` are all related to our APT campaign, the other types and `EternalSafety::ViolationTx2Cmd` produce high volumes of unrelated alerts and are responsible for a large increase of alerts from MIN to FULL.

Summing up, the MIN configuration produced Zeek alerts for the second stage trojan download, the lateral movement via PsExec, and the data exfiltration. It missed the initial RCE and the Cosmic Duke C2/C&C communication, which thus cannot be contained in any further processing. The FULL configuration improves the detection of lateral movement via additional alerts from BZAR, while the EternalSafety package is able to produce a few alerts related to the RCE among a large number of false positives.

Meta-alerts and single alerts After correlation via GAC, we obtained clustered meta-alerts and unclustered that both serve as input for the APT contextualization. Table 5.11 shows which meta-alerts and unclustered alerts are related to IDS2018-APT for both configurations.

For the MIN configuration GAC produced 119 meta-alerts and 4 432 unclustered alerts. Out of these total 4 551 alerts, only three single alerts are related to IDS2018-APT. This reinforces the expectation, that alerts caused by APT activity are not clustered by traditional alert correlation algorithms. Given these results we expect an accurate

Source	Alert Type	IDS2018-APT-**MIN** # Alerts	IDS2018-APT-**MIN** APT related	IDS2018-APT-**MIN** Ratio	IDS2018-APT-**FULL** # Alerts	IDS2018-APT-**FULL** APT related	IDS2018-APT-**FULL** Ratio
Zeek	Conn::Retransmission_Inconsistency	1 171	0	0.00	1 171	0	0.00
	SSL::Weak_Key	120	0	0.00	120	0	0.00
Organization-specific	Org::Stalled_HTTP_Connection	4 976	0	0.00	4 976	0	0.00
	Org::HTTP_Windows_Executable_Dl	6	0	0.00	6	0	0.00
	Org::NON_HTTP_Windows_Executable_Dl	4	1	0.25	4	1	0.25
	Org::SMB_Executable_File_Transfer	1	1	1.00	1	1	1.00
	Org::Javascript_Web_Injection_URI	336	0	0.00	336	0	0.00
	Org::SQL_Web_Injection_URI	78	0	0.00	78	0	0.00
	Org::Web_Login_Guessing	14	0	0.00	14	0	0.00
	Org::Large_Outgoing_Tx	5 772	0	0.00	5 772	0	0.00
	Org::Multiple_Large_Outgoing_Tx	187	0	0.00	187	0	0.00
	Org::Very_Large_Outgoing_Tx	10	1	0.10	10	1	0.10
BZAR	ATTACK::Execution	—	—	—	2	2	1.00
	ATTACK::Lateral_Movement	—	—	—	4	4	1.00
	ATTACK::Lateral_Movement_Extracted_File	—	—	—	1	1	1.00
EternalSynergy	EternalSafety::DoublePulsar	—	—	—	1	1	1.00
	EternalSafety::EternalBlue	—	—	—	53	0	0.00
	EternalSafety::EternalSynergy	—	—	—	1	1	1.00
	EternalSafety::ViolationCmd	—	—	—	1 389	0	0.00
	EternalSafety::ViolationNtRename	—	—	—	8 731	0	0.00
	EternalSafety::ViolationPidMid	—	—	—	6 133	0	0.00
	EternalSafety::ViolationTx2Cmd	—	—	—	408 686	1	0.000002
	Signatures::Sensitive_Signature	—	—	—	8 731	0	0.00
	Total	12 675	3	0.000237	446 407	13	0.00000245

Table 5.10: Results: Ground truth in Zeek alerts for IDS2018-APT

	IDS2018-APT-**MIN**			IDS2018-APT-**FULL**		
Attack Step	Type	Source	Target	Type	Source	Target
EternalRomance	—	—	—	Meta	1.1.13.37 + 7 unrelated	172.31.64.67
2nd stage trojan	Single	172.31.64.67	12.34.12.34	Single	172.31.64.67	12.34.12.34
Cosmic Duke	—	—	—	—	—	—
PsExec via SMB	Single	172.31.64.67	172.31.69.20	Single	172.31.64.67	172.31.69.20
Data exfiltration	Single	172.31.69.20	1.1.15.57	Single	172.31.69.20	1.1.15.57

TABLE 5.11: Results: Ground truth in clustered alerts for IDS2018-APT

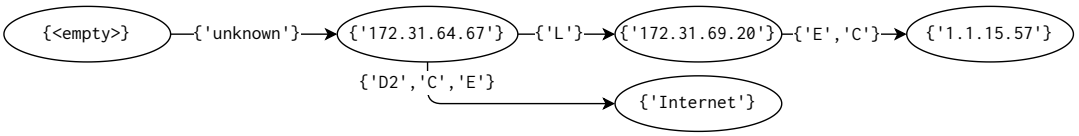


FIGURE 5.23: Result: APT scenario graph for IDS2018-APT-MIN

contextualization result as all alerts contain single IP addresses for as source and destination. In the FULL configuration GAC generated 10 713 meta-alerts and 39 649 unclustered alerts. Compared to the unmodified dataset these numbers are in the expected range. Out of the four relevant alerts, we see three single alerts for the same attack steps as in MIN plus an additional alert for the initial RCE as expected from the Zeek alerts show previously. However, the alert that captured this step is a meta-alert clustered by GAC and contains multiple source IP addresses. This has two implications: First, the resulting APT scenario graph will likely contain all eight IP addresses as the algorithm cannot split this set without additional intelligence. Second, this example shows that meta-alerts may also carry (partial) information related to APTs. While the results from the MIN configuration may indicate that it is sufficient to only investigate unclustered alerts, this does not hold in the FULL configuration. Thus, we expect a similar contextualization result compared to MIN with more details about the initial point of infection.

APT scenario graphs For the MIN configuration, the APT Contextualization yielded 611 distinct APT scenario graphs. From the original set of 12 675 alerts, this implies a reduction to **4.82%**. Among the result set is one graph that describes our APT campaign. Figure 5.23 shows the APT scenario graph that was generated from the alerts of IDS2018-APT. As mentioned before the initial infection via *EternalRomance* and the Cosmic Duke C2/C&C did not generate an alert and are thus not included in the graph. Most stages are labeled correctly with IP addresses and APT stages. The only imprecision relates to the second stage trojan download: the edge is labeled ambiguously with `[E, D2, C]` and the corresponding node not only contains the target IP address 12.34.12.34 but the label "Internet". The stage mismatch is expected as our approach derives potential stages from network direction and can not reduce the set of stages further without additional context for outgoing connections to the Internet. The label is the result of imperfect meta-alert generation as the algorithm grouped multiple Internet IP addresses and thus produced that label. However, the scenario references all used meta-alerts and unclustered alerts via a unique identifier.

The FULL configuration produced an alert for the initial RCE that was missed in MIN. Ideally, the produced APT scenario graph should thus correctly identify the first node in the chain that is labeled as with `{<empty>}` in Figure 5.23. The contextualization

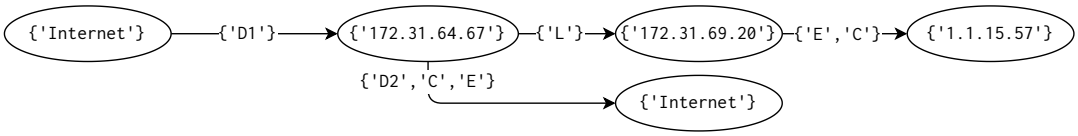


FIGURE 5.24: Result: APT scenario graph for IDS2018-APT-FULL

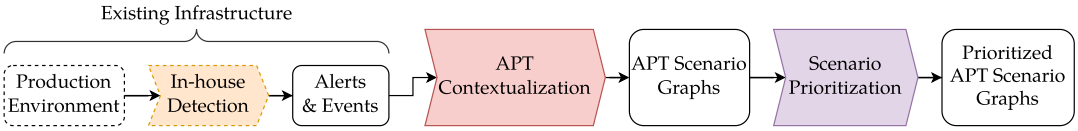


FIGURE 5.25: Experimental Setup: Enterprise datasets

yielded 686 distinct APT scenario graphs—a reduction to **0.15%** of the total alert set. Figure 5.24 shows the one that closely resembles our APT campaign. The new APT scenario graph matches our campaign with similar precision than the one from MIN. The initial RCE is picked up and added to the scenario. However, as multiple alerts relate to incoming connections to 172.31.64.67, the node is labeled with "Internet". The referenced alerts indeed contain the one related to the RCE. Overall the FULL scenario matches the campaign more closely as expected from the generated alerts. Human analysts can use the graph to further investigate the referenced alerts to confirm APT activity.

5.3.6.3 Enterprise Datasets

This section contains the evaluation of the extended approach for APT contextualization developed in the supervised Bachelor's thesis [Lau22] (**APT Contextualizer v2**) based on the two variants of the associated enterprise dataset and the embedded synthetic APT campaign.

Experimental Setup Due to the amount of private information in the enterprise dataset, evaluation of **APT Contextualizer v2** was performed on-site by the student with only aggregated results being available due to compliance reasons. Additionally, the evaluation started on alert-level as this was the only available data source in the production environment of the medium-sized managed security provider. However, the previous evaluation already demonstrated, that our approach is able to cover the whole detection pipeline from pcap to APT scenario graphs. Figure 5.25 gives an overview about the evaluation pipeline that was used. The existing infrastructure already generates alerts from the production environment via a custom in-house detection pipeline that our approach treats as opaque. The alerts are stored in a SIEM from where our approach retrieves them for the contextualization process. After APT scenarios are generated, they are prioritized based on the scoring mechanism described in Section 5.3.5.2 to obtain the final set of prioritized scenario graphs.

Experiment III: Volume Reduction Similar to the first experiment, we first analyze the volume reduction achieved by **APT Contextualizer v2**. We chose to perform this experiment on the datasets that include the six embedded alerts from the synthetic APT campaign, as the system works on alert-level anyway and the difference in alerts is negligible. Both dataset variants are processed by our pipeline and the number of resulting APT scenario graphs is compared with the size of the respective original alert sets.

	Baseline			Stage Mapping	
	v1	v2 Alerts	+Events	v2 Alerts	+Events
Alerts	— 17 726 false positives + 6 APT-related —				
Infection Trees	146	165	175	147	213
APT Scenarios	276	145	192	171	211
- Average Nodes	3.01	3.90	4.28	2.10	2.10
- Average Edges	2.01	2.90	3.28	1.90	2.90
Reduction [%]	1.56	0.82	1.08	0.96	1.19
Scenarios per Day	9.20	4.83	6.40	5.70	7.03

TABLE 5.12: Results: Volume reduction for Enterprise-Small

Table 5.12 shows the results for Enterprise-Small. The first column contains the results of **APT Contextualizer v1** applied to the small 30-day dataset (Enterprise-Small). The total number of APT scenarios is moderate with 146 which equals to 9.2 scenarios per day. The reduction of 9.20% of the original alert set is larger than our previously obtained results on CSE-CIC-IDS2018 [SHG18]. The second column contain the results of **APT Contextualizer v2** in the base configuration that only relies on the network direction to map APT stages to alerts. We already see an improvement in both total scenario obtained (145) and average number of graph elements compared to the **APT Contextualizer v1**. The third column also adds event enrichment via graph connectivity as described in Section 5.3.5.1. As expected this increases the number of generated APT scenario graphs again to 192 total or 6.4 per day. The last two columns show results for enabled stage mapping, i. e., usage of attached stage labels on alerts for improved precision of assigned stages. The results are comparable to the **APT Contextualizer v2** baseline with slightly more generated scenarios. However, the number of average graph elements is slightly surprising with the lowest among the table. This is likely the result of some short graphs consisting of mostly host alerts.

Table 5.13 contains the same configurations for Enterprise-Large, the 90-day dataset. Due to the increased number of alerts, the total amounts of scenarios is rising as expected. While the reduction is still decent with 6.69% for **APT Contextualizer v1** and 3.30–3.59% for **APT Contextualizer v2**, the high number of about 54k alerts results in 40 and 20–21 scenarios per day. This might strain a small- to medium-sized SOC. It is also interesting to note, that the overall average count of graph elements drops compared to Enterprise-Small, indicating that this dataset contains more short APT scenarios. This is caused by differing traffic patterns between the months of capture. Overall, the results for volume reduction are promising for both datasets and highlight the improvements made by **APT Contextualizer v2**. The number of APT scenarios is consistently lower compared to **APT Contextualizer v1** with similar or even higher complexity (as indicated by the average number of graph elements). The next experiment investigates if these improvements impacted the reconstruction performance.

Experiment IV: Campaign Reconstruction While volume reduction is essential to unburden SOC analysts, our approach would be significantly less valuable if APT campaigns cannot be accurately reconstructed. To evaluate this, the resulting scenario graphs are manually analyzed to find our injected, synthetic APT campaign, that we described in Section 5.3.6.1. As alerts for all attack steps were injected into the

	Baseline			Stage Mapping	
	v1	v2 Alerts	+Events	v2 Alerts	+Events
Alerts	— 54 220 false positives + 6 APT-related —				
Infection Trees	1 888	1 856	2 029	1 859	2 029
APT Scenarios	3 626	1 888	1 936	1 790	1 945
- Average Nodes	3.05	3.30	3.30	2.01	2.01
- Average Edges	2.05	3.20	3.21	2.00	2.00
Reduction [%]	6.69	3.48	3.57	3.30	3.59
Scenarios per Day	40.31	20.98	21.51	19.89	21.61

Table 5.13: Results: Volume reduction for Enterprise-Large

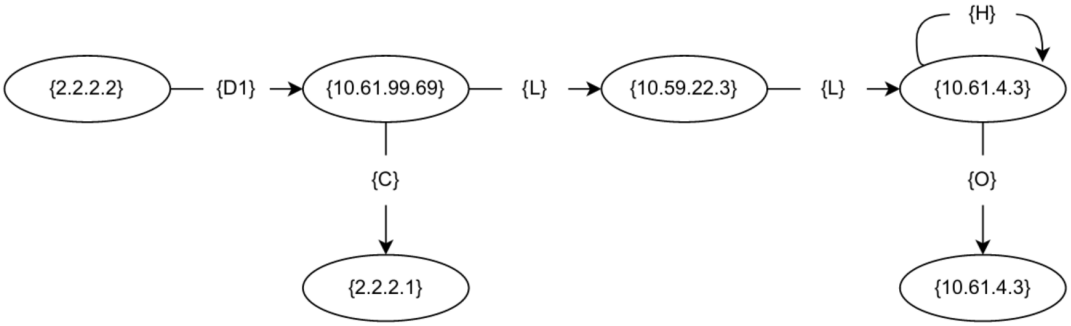


Figure 5.26: Result: APT scenario graph in the enterprise dataset

timeframe of the datasets present in both Enterprise-Small and Enterprise-Large, we expect a fully reconstructed scenario without any missing stages to be found in all experiments. This is exactly what happened and Figure 5.26 shows the reconstructed APT campaign. We see, that the complete campaign is reconstructed including the host-based reflexive edge and the last *objective* stage which was also detected on host-level. Combined with the results for volume reduction shown in the previous experiment, we can conclude, that **APT Contextualizer v2** is capable of extracting embedded information about potential APT campaign activity from a large set of alerts resulting in detailed visual representations of them.

Experiment V: Scenario Filtering and Prioritization Another important improvement made by **APT Contextualizer v2** is the ability to filter the resulting set of APT scenario graphs to reduce the investigation load on SOC analysts even further. The current implementation offers two *filtering strategies* to achieve this that can also be used on conjunction if desired:

- **Length-based filtering** improves upon the basic filtering feature already present in **APT Contextualizer v1**. Instead of simply filtering all graphs with less than two nodes (including potential *unknown infection start* nodes), the new algorithm prunes all scenarios with less that two stage transition edges. This results in chains of host-based alerts (without any other involved hosts) to be deleted.
- **Priority-based filtering** uses the *priority score* as described in Section 5.3.5.2 and filters all scenarios that score less than a configurable threshold. As the score aims to represent an overall threat level, this strategy is a good fit for SOCs that are strained for resources.

	Baseline/None		Length-based		Priority-based (40)	
	Alerts	+Events	Alerts	+Events	Alerts	+Events
Alerts	— 17 726 false positives + 6 APT-related —					
APT Scenarios	171	211	4	15	148	178
- Average Nodes	2.10	2.10	4.30	3.89	2.11	3.31
- Average Edges	1.90	2.90	4.20	3.94	2.00	3.21
Reduction [%]	0.96	1.19	0.02	0.08	0.83	1.00
Scenarios per Day	5.70	7.03	0.13	0.50	4.93	5.93

Table 5.14: Results: Scenario filtering for Enterprise-Small

	Stage Mapping		Length-based		Priority-based (40)	
	Alerts	+Events	Alerts	+Events	Alerts	+Events
Alerts	— 54 220 false positives + 6 APT-related —					
APT Scenarios	1 790	1 945	16	32	1 328	1 368
- Average Nodes	2.01	2.01	2.18	2.11	2.01	2.01
- Average Edges	2.00	2.00	2.18	2.13	2.00	2.00
Reduction [%]	3.30	3.59	0.03	0.06	2.44	2.52
Scenarios per Day	19.89	21.61	0.17	0.35	14.76	15.20

Table 5.15: Results: Scenario filtering for Enterprise-Large

We evaluate both strategies on both datasets. Table 5.14 contains the results for the 30-day Enterprise-Small dataset. The left two columns are the same results with from the volume reduction experiment with stage mapping enabled. As both filtering strategies rely on precise stage assignments for alerts (which is achieved by stage mapping), these results serve as a baseline. The following columns show the results for both filtering strategies with optional event enrichment (40 was chosen as the threshold for priority-based filtering). It is important to note, that our synthetic APT campaign was never filtered.

We see, that length-based filtering performs exceptionally well, reducing the total number of APT scenario graphs to 4 when only alerts are considered and 15 with enabled event enrichment. This implies a massive reduction to 0.13%(**!**) and 0.50% of the total alert set respectively. As expected, the average number of graph elements is thus comparatively high with 3.89–4.30 nodes and 3.94–4.20 edges. This indicates that the filtering worked as intended and retained complex campaigns. However, we gave to consider, that our scenario was injected with "perfect visibility" assuming alerts are actually generated for all relevant attack steps. The results show, that length-based filtering is highly efficient when such long and complex scenarios need to be reconstructed. For other scenarios, e. g., incomplete or in-progress campaigns, priority-based filtering is a better choice. With our moderate threshold of 40, the strategy filtered 23–33 APT scenarios compared to the baseline resulting in 148 total scenarios derived from just the alert set and 178 scenarios with added event enrichment. It is also important to keep in mind that both the priority score calculation as well as the threshold can be changed to adapt **APT Contextualizer v2** to a custom environment. Furthermore, the remaining graphs can still be prioritized via the score to further improve SOC efficiency.

Results for Enterprise-Large are shown in Table 5.15. As expected, length-based filtering performs well again achieving reductions to 0.17–0.35%. However, the average number of graph elements drops significantly to 2.11–2.18 nodes and 2.13–2.18 edges which is slightly surprising. Nonetheless, our synthetic APT campaign is still retained. For priority-based filtering the improvements compared to the baseline are nearly the same as for the small dataset and we see scenarios per day reduced from 19.89–21.16 to 14.76–15.20. This indicates a manageable investigation load for the SOC especially compared to the initial baseline set by **APT Contextualizer v1** in the volume reduction experiment of 40.31 APT scenario graphs per day.

In summary, our evaluation indicates that our approach for contextualization of APT campaigns based on kill chain state machines works as intended. (*i*) it significantly reduces the overall set of alerts to be investigated by two to three orders of magnitude and (*ii*) it can detect and contextualize complex attacks. Although not all nodes are labeled perfectly, large parts of our APT campaign are visualized. While our APT campaign is obviously not representative of all potential APT campaigns it does include zone movement that is characteristic of typical multi-stage attacks. The movement of the attacker across different zones should result in longer chains in our APT scenario graphs and thus reveal the campaign. **APT Contextualizer v2** further improves upon the original implementation in several ways and adds features to prioritize and filter the result set of APT scenario graphs. Our results indicate that SOC analysts can be significantly unburdened as the compact representation of potential APT campaigns combined with prioritization measures helps to quickly reveal connected APT activity.

5.3.6.4 Requirement Comparison

Based on the description and evaluation results, we can now compare our approach for APT contextualization based on KCSMs with our requirements formulated in Section 3.1. *R1: Accuracy* is fulfilled as our approach achieves a massive reduction in alert volume while retaining essential information about the potential APT campaign embedded in the alert set. While the absolute accuracy, i. e., number of relevant scenarios compared to the overall number of scenarios, is not perfect, especially **APT Contextualizer v2** achieves good results while balancing the trade-off between over-filtering and low accuracy. *R2: Explainability* is also fully met as the APT scenario graphs compactly visualize the additional information revealed of our contextualization process. There are two factors influencing *R3: Low overhead*: (*i*) the little restrictions imposed on the input alert set (tagged or untagged) and (*ii*) the medium amount of computation required for the contextualization process. As a consequence, we mark this requirements as partially checked. Due to the missing runtime measurements, *R4: Scalability* cannot be reliably rated and is left as unknown. *R5: Security* was not an explicit design goal of our approach as it is only tangentially relevant. It is thus left as neutral. We mark *R6: Privacy* as partially fulfilled as our approach requires only high-level information in its input alerts and can thus work with potentially redacted alerts (as long as IP addresses are left intact). Lastly, *R7: Deploy- & Maintainability* is fully met as our approach is highly flexible in which alert types can be consumed and should thus be comfortably integrable in most enterprise environments.

5.3.7 Summary

This section presented an approach for APT campaign reconstruction based on a novel Kill Chain State Machine (KCSM), an actionable formalization of the UKC [Pol21],

that generates compact representations of potential APT campaigns to answer our formulated research questions **RQ5** and **RQ6**:

RQ5 *How can established kill chain-based APT models be leveraged for campaign detection with only minimal assumptions about the lower-level alert set?*

RQ6 *Which level of volume reduction can be achieved during APT campaign reconstruction to lessen the impact of alert fatigue on SOC analysts?*

Our algorithm ingests both regular untagged alerts and stage-specific alerts and links them according to pre- and post-conditions of the KCSM to obtain APT scenario graphs. The original approach was implemented as a Python-based prototype and evaluated on the CSE-CIC-IDS2018 dataset [SHG18] that we enhanced with a synthetic APT campaign. This "end-to-end" evaluation starts on pcap level and results in APT scenario graphs and thus demonstrates that our approach is applicable in real-world settings. Additionally, the implementation was improved to better handle host-based stages and improved prioritization measures. This version was evaluated in a real-world enterprise setting with an injected APT campaign to further show how our approach is capable of real-world usage. Results across both evaluations indicate strong performance in both (*i*) reducing the overall alert volume to 0.96–3.59% of the original alert set (depending on the exact configuration and scenario) and (*ii*) correctly reconstructing the APT. Furthermore, we show how the configurable prioritization feature can further improve the final result set and thus saving valuable time of SOC analysts. Overall, our experiments and obtained results across two datasets provide answers to the research questions: (*i*) The successful reconstruction of both synthetic APT campaigns indicates that our formalization of established kill chain-based models to the KCSM enables our approach to detect APT campaign activity. (*ii*) The significant volume reduction achieved in both the unmodified datasets and with the injected APT campaigns indicates that we can obtain volume reductions to 2–3 order of magnitude while preserving the contextual information of potential APT campaign activity.

5.4 Summary

APT detection approaches extract higher-level meaning from the low-level alerts and events obtained through security monitoring. APT campaigns impose extra challenges on this process due to their extended stay in the network and their overall sophistication. This chapter introduced three core contributions of this thesis that advance the state-of-the-art in the area of APT detection. A stage-specific approach for lateral movement detection is followed by a state machine-based approach for APT detection as well as a concept to regain explainability for graph-based AI detection approaches. Together, these three approaches offers improvements throughout the APT detection process in both forensic and detection settings.

The approach presented in Section 5.1 aims to reveal hosts that have been compromised by lateral movement during an APT campaign based on an incomplete set of alerts or IoCs. A formal model describing both the underlying network and three potential attacker classes lays the foundation for the algorithm. It is based on the notion of Criticality, a node feature that combines host vulnerability and importance to mark hosts that are most likely to become compromised. We then designed an abstract algorithm that approximates the set of infected nodes without specifying a concrete implementation to obtain nodes and implemented two variants based on k-shortest paths and random walks. We evaluate our approach on synthetic datasets consisting of

(*i*) randomized network topologies according to our formal model that resemble real-world organization networks and (*ii*) generated attacks that are reduced to incomplete alert sets based on our three attacker classes. Our results indicate that k-shortest paths consistently outperform random walks even for attackers that strongly deviate from our proposed attacker classes and are thus the recommended implementation.

Section 5.2 introduced a concept to restore explainability for graph-based anomaly detection approaches by leveraging a variation of the established XAI technique permutation importance. We can identify graph elements that are likely part of the anomalous part of the graph by systematically modifying the original graph and observing the changing anomaly score that are reported by the anomaly detection approach in question. While we apply and evaluate our prototype implementation in the domain of APT detection based on system provenance data, the concept is transferable to other graph-based whole-graph classification problems as the anomaly detection function is treated as a black box without any further assumptions about the underlying AI model. We evaluate our approach on two popular system provenance datasets, namely StreamSpot [MMA16] and DARPA TC Engagement 3 [DAR18] using UNICORN [Han+20] as our anomaly detection approach to augment. Our results indicate that our approach is able to identify nodes that are likely part of the anomaly using the node removal strategy. While the fidelity of the datasets is insufficient to fully validate our findings, we successfully tested our results for statistical significance using the Wilcoxon rank-sum test [Wil92].

The approach presented in Section 5.3 aims to reconstruct APT campaign activity based on a Kill Chain State Machine (KCSM) derived from the UKC [Pol21] by representing general campaign progress as states and kill chain stages as transitions. The model can be used to track APT campaign progress throughout the network and also keeps track of potentially infected hosts (via their IP addresses). The approach contains an algorithm to map potential APT stages to any network-based alert and is this independent from the underlying alerting. However, it can also ingest alerts that are specifically tagged with a set of potential APT stages and include them in the reconstruction process. Next to the network-based prototype, an extension, that features several improvements including support for a subset of host-based stages, is also presented and evaluated. The algorithm results in APT scenario graphs that offer a compact visual representation of reconstructed potential APT campaigns and can support SOC analysts in incident response and mitigation measures. Our results indicate that both implementations of our approach are able to (*i*) significantly reduce the overall alert volume to 0.96–3.59% of the original size and (*ii*) compactly represent connected malicious activity from potential APT campaigns in the network.

Together with the approaches for improved visibility introduced in the previous chapter, this chapter presented the contributions of this thesis. The five presented approaches improve several key areas required for APT detection from low-level visibility to whole campaign detection. The approaches can be used individually or in conjunction. One example deployment could leverage our system for visibility into TLS encrypted traffic to obtain a baseline alert set by via a NMS/IDS. This baseline is then further augmented with stage-specific alerts from from both the analysis of brute-force login attempts (tagged with reconnaissance) and our algorithm to reconstruct lateral movement. All three types of alerts can then be processed by our APT reconstruction via the KCSM to obtain a holistic overview about the current potential campaigns in the network. This flexibility marks one additional benefit of

the approaches presented in this thesis. The following chapter summarizes this work once more and provides an outlook on remaining open questions and future work.

6 Conclusion

This chapter briefly summarizes the open challenges for the detection of advanced persistent threat (APT) campaigns in enterprise networks, the contributions towards these challenges made in this thesis as well as potential directions for future research. Section 6.1 recaps the requirements for APT detection established in this thesis and how they relate to relevant prior work and the contributions made towards improved security monitoring and APT detection. Section 6.2 concludes this thesis with remaining problems and open challenges for future work in this domain.

6.1 Summary of Contributions

The emerging hazard from APTs poses a challenge for large organizations, critical infrastructure, and governments alike. Detection of APT activity is hindered by their stealthiness and overall sophistication level, such that adversaries often remain in networks for extended periods of time. The process is hindered two major factors: (*i*) limited visibility due to insufficient *security monitoring* such that key APT activity remains undetected and (*ii*) *alert fatigue* of security operations center (SOC) analysts, who are overwhelmed by the high-volume alert stream due to a lack of accurate detection and reconstruction approaches. Both high-quality security monitoring that reliably obtains indicators of even stealthy activity and sophisticated security analytics that filters and correlates the large amounts of generated alerts down to a manageable level are essential for successful detection of APT attacks.

This thesis formulates requirements for APT detection in enterprise networks and discusses relevant literature across the whole detection process according to them. The survey of prior work in the respective areas reveals multiple open challenges for both security monitoring and security analytics which this thesis aims to address. On event-level, adoption of encryption for network traffic, e. g., via TLS, has limited the visibility network monitoring systems (NMSs) can obtain into application-level payloads. Existing proposals exhibit several drawbacks including security issues, privacy concerns for users and lack of wide-spread adoption among others. Additionally, the potential of APT reconnaissance through brute-force login attempts is not covered by existing approaches as they focus on the classification task between benign and malicious logins. On alert-level, there are some promising approaches to detect APT activity through anomaly detection on system provenance data that achieve generally good detection performance. However, due to the complexity of the underlying models, results are not easily interpretable and thus fail to deliver additional attack context. This hinders effective incident response as security experts need to further investigate details about the attack to estimate their impacts Alert correlation is another relevant research area, as it also aims at reducing the overall alert volume, e. g., by clustering related alerts. However, approaches from this area are usually not directly applicable to APT campaigns as they result in few alerts only (if any). This hypothesis has also been confirmed by our analysis of selected works

in this area as the approaches fail to achieve good accuracy. Approaches for APT stage detection are valuable tools to obtain indicators for APT activity. Our analysis of three key APT stages revealed that no approach manages to achieve high accuracy while also being deployable in enterprise contexts. APT campaign reconstruction is the most promising area of research related to the overall goals of this thesis. The survey of the relevant literature revealed that no approach achieves good accuracy without relying on system provenance data, which is still hard to reliably capture and process in enterprise environments. Overall, our literature survey of APT detection concluded that there are many promising approaches but also several unresolved challenges. This thesis thus aims to propose mechanisms that address these gaps for both enhanced security monitoring and detection of entire APT campaigns.

The first contribution to enhance security monitoring is offered in form of our approach for *passive TLS decryption via cooperating endhosts*. This helps to regain visibility lost by the use of TLS encryption. The approach significantly improves upon the current state of the art, in which TLS-intercepting proxy servers are used, by keeping the integrity properties of the TLS connection intact. This is achieved by cooperatively forwarding key material from the network stack of the endhosts to the trusted NMS where it can be used for decryption. We implemented the approach as a prototype module for the Zeek NMS [Pax99; Zee22] and a Python-based daemon that runs on the endhost. Conceptually, our approach requires less computational resources compared to systems that rely on active interception of the TLS connection, such as proxy servers, as these systems need to both decrypt and encrypt all payload bytes twice. As our NMS only needs to passively decrypt payloads for analysis (and can even skip "uninteresting" traffic), it is less CPU intensive. Furthermore, it supports selective decryption (in applicable scenarios) and thus allow users of the endhosts to retain conceptually guaranteed privacy for selected connections. Our evaluation results indicate that the overhead induced by performing decryption in the NMS is acceptable at around 2.5 times the original runtime. Considering the resources that can be saved by not performing costly active interception, our approach should not result in overall increased resource requirements. Additionally, we evaluate the impact of delay resulting from the transfer of key material via the network on the decryption performance. The results indicate that a small traffic buffer of about 40ms should be enough to decrypt 99.9% of observed connections in our test scenario. Lastly, our prototype gained traction in the Zeek community, prompting us to further improve it. As a result, our modifications have been upstreamed to mainline and are available starting from Zeek version 5.0.0.

Another improvement towards better external visibility on APT activity is contributed by our novel approach for *characterization of brute-force attackers*. By using both established metrics from literature and two novel metrics proposed by us, SOC analysts can make estimates about attacker behavior that are helpful when triaging login attempts on their publicly accessible hosts. Single metrics or metric combinations can be used to cluster attacking IPs and thus reveal potential collusion. Also, both standalone metrics or combinations thereof can offer insights about attacker behavior and likely sophistication level. They can thus be leveraged to prioritize attacks and attack clusters for analysis. Our evaluation on a real-world dataset captured by our Honeygrove [Hon+22] honeypot indicates that the analysis process based on our set of metrics can provide strong indicators for collusion for otherwise unrelated IP addresses and produce useful clusters that speed up the following analysis.

Next to these two enhancements for security monitoring, this thesis also proposes

three mechanisms to effectively support SOC analysts in coping with alert fatigue via sophisticated security analytics approaches. The following contributions all aim to extract higher-order meaning from the existing alert set by (*i*) estimating impact of lateral movement for improved incident response, (*ii*) adding missing context about detected anomalies cause by APT attacks and (*iii*) reconstructing entire APT campaigns.

The first contribution in the area of security analytics is a *stage-specific approach to reconstruct lateral movement* from an incomplete alert set. The algorithm identifies hosts that were likely compromised as during lateral movement of a detected APT attack based on an underlying model that encompasses both host security properties and already attributed alerts. The resulting ordered lists of hosts speeds up incident response as security experts can prioritize hosts for cleanup procedures. We implemented two variants of our approach, one based on k-shortest paths as well as one based on biased random walks, and evaluated both on a synthetic dataset. Our results indicate that the variant based on k-shortest paths strongly outperforms the alternative across all three attacker classes. In addition to the experiments with idealized attackers, we also evaluated the influence of attacker deviation on the detection performance. Similar to the idealized attackers detection performance is best for insider attackers with a true positive rate (TPR) of about 90% at even 80% deviation. Overall, we recommend to configure the approach with the result set size parameter τ set to 0.2 to achieve best results across all three attacker classes.

Next, this thesis presents an approach to *restore explainability for graph-based anomaly detection approaches* that adds essential context about a detected attack that is typically unavailable due to the complexity of the underlying normality models. In the case of APT detection this loss is especially important as SOC analysts rely on detailed information about why a certain system was deemed compromised to inform mitigation and recovery processes. Our approach addresses this problem by treating the anomaly detection function as a blackbox, systematically permuting the input data, and observing the generated anomaly scores. The permutation choices are governed by modular strategies of which we provide two: one based on node removal as well as another one based on edge removal. While our approach is designed for the specific domain of anomaly-based APT detection on data provenance graphs, our proposal is not strongly tied to the domain and can be used for other whole-graph classification problems as well. In fact, the two presented permutation strategies do not rely on semantic information from the graph and only leverage structural data also present in graphs from other domains. Our evaluation uses UNICORN [Han+20] as the anomaly detection approach and is based on the StreamSpot [MMA16] and Defense Advanced Research Projects Agency (DARPA) Transparent Computing [DAR18] datasets. While the fidelity of the datasets did not allow us to fully confirm our identified abnormal graph elements, we showed how our area under baseline (AUB) metric can be used to quantify the "abnormality" of the graph in question. We observed a difference in AUB distribution between benign and abnormal StreamSpot graphs indicating higher median values for the abnormal graphs (i. e., the ones that contain an attack). This aligns with our hypothesis that the AUB quantifies the abnormality, thus resulting in higher (median) values for attack graphs. Additionally, we confirmed the statistical significance of these results using the Wilcoxon rank-sum test [Wil92].

The last contribution is an approach for *APT campaign reconstruction based on the novel concept of a Kill Chain State Machine* (*KCSM*). By deriving a state machine from the most comprehensive kill chain model to date, the unified kill chain (UKC) [Pol21], we make

the abstract concept of kill chains more actionable. The resulting algorithm ingests alerts (that may optionally be clustered in a preprocessing step) and traces potential APT campaign activity across all hosts in the network. After potential APT stages for an alert are derived based on network direction, the algorithm links related alerts and finds connected infection chains based on the pre- and post-conditions of the KCSM. The resulting artifacts are called APT scenario graphs and offer a compact overview about the reconstructed campaigns. SOC analysts can then leverage this information for triage or further threat hunting activities. In addition to the base algorithm described above, we improved out prototype implementation to (*i*) better handle alerts that are pre-tagged with APT stage information and (*ii*) filter and prioritize the result set to better utilize the limited analyst time. The resulting implementation is more modular, offers slightly increased performance and can be adapted to new environments easily. Our evaluation is based on two scenarios: (*i*) We evaluate the basic algorithm "end-to-end", e. g., from pcap level up to the final graphs, on an established intrusion detection dataset that we enrich with a carefully crafted synthetic APT campaign. Our results indicate that our APT campaign is fully reconstructed as long as alerts are available in the underlying alert set. This demonstrates that our algorithm is able to trace typical APT campaigns and highlights once again the importance of good-quality security monitoring as enhanced by the first two contribution of this thesis. Additionally, our approach is capable of reducing the alert volume drastically down to 0.15%(!) of the original alert set. While our approach still produces some false-positives even when no APT activity is present, this reduction and the added context enables SOC analysts to investigate the remaining scenarios more effectively compared to the singular alerts before. (*ii*) We deploy the extended approach in a medium-size security provider to evaluate the detection performance in a real-world scenario. Similar to the first experiment, we also inject a synthetic APT campaign based on analyst experience. The results indicate that the extended version is capable of tracing APT campaigns in real-world scenarios that include host-based alerts while maintaining volume reduction ratios between 0.96–3.59%. Additionally, a comparison of added prioritization methods highlights how SOC analysts can be supported even after scenarios are generated resulting in reduction rates of down to 0.02% (4 generated scenarios from 17 732 total alerts) in extreme cases. Although, this specific configuration is not applicable to all organizations, it demonstrates the flexibility of our approach in ideal settings.

In summary, this thesis presents two approaches that enhance security monitoring to obtain alerts of both higher quality and quantity as well as three approaches for improved security analytics to lower the burden of the large amounts of alerts on SOC analysts. The five contributions in total each address a pressing need in their respective area and thus support the overall goal of APT detection. They can be used individually or in conjunction to improve the detection capabilities of any SOC protecting an enterprise network. However, some areas still need further improvements as the next section describes.

6.2 Future Work and Outlook

APT detection remains an extremely challenging task. While the contributions presented in this thesis address important gaps in the state of the art, several research areas retain further questions that remain open. This section briefly highlights some of these problems that arise from either limitations of the approaches presented in this thesis or remain from challenges that have not been addressed yet.

The approach for passive TLS decryption via cooperative key sharing from endhosts (that we present in Section 4.1) is designed to conceptually support *selective decryption*, i. e., allowing users to withhold key material and thus retain some privacy for selected connections. We argued that this can improve user trust and willingness to participate in the overall system in certain scenarios. Future research is required to evaluate *the applicability of selective decryption in different real-world scenarios* such as a university campus or a mid-sized company that wants to promote employee privacy. A quantitative user study could then provide insights in how the feature is accepted by real users and how it influences their acceptance of the overall security monitoring.

Accurate mapping of security-relevant properties of hosts and other assets is a complex task that should also be improved by future work. As the network constantly evolves by either hardware changes and host churn or changing software versions that fix and introduce new vulnerabilities, a comprehensive database that details the security state of the network and all assets can provide substantial benefits for defense. While tools exist for sub-areas like *nessus* [Ten22] for network scanning and asset discovery, vulnerability databases like CVSS [For15], or *nmap* [Ln22] to detect software versions, future work should aim to *combine and reconcile the results* to obtain a unified overview about the security state of the network. Such a database could also serve as the input to our approach for lateral movement reconstruction (that we introduce in Section 5.1).

The approach to restore explainability to complex graph-based models for APT detection (that we describe in Section 5.2) already shows promising results. Although the current evaluation can not sufficiently prove that our approach correctly identifies the correct nodes resulting from the detected attack, the results based on our AUB metric provide good indications for subsets of nodes that are likely part of the anomaly. Due to missing accurate ground truth, the current datasets are unfit to obtain ideal results. Future work can improve this by either generating new datasets of higher granularity such that individual graph elements of anomalies can be identified or by manually annotating the existing datasets based on expert analysis. Furthermore, more complex permutation strategies which leverage the semantic information encoded into the nodes should be investigated to further improve results in the context of APT detection.

Another opportunity for future research in the APT context lies in the reliable detection of host-based APT stages (or chains thereof) that can be used as part of a larger reconstruction algorithm. While there are several approaches that tackle network-based stages, the host-based stages are somewhat underrepresented in the state of the art. The high fidelity of system-level host data is appealing for artificial intelligence (AI)-based approaches that rely on a broad training set. As stage-specific detection approaches are usually not investigated on their own but in context with other related alerts, a novel approach could leverage complex AI models that are not interpretable as the resulting stage label offers enough context for the reconstruction compared to a anomaly detection-based approach that treats any anomaly as an attack.

Our presented approach for APT contextualization based on the KCSM (that we introduce in Section 5.3) produces good results for alert- and event-data data obtained in the network. Future work should extend the reconstruction algorithm to efficiently *integrate external cyber threat intelligence* (*CTI*) to further improve accuracy of the generated scenarios. CTI is already a valuable asset in APT detection that is currently mostly used for either automated rule-based blocking or manually consulted during investigations. This integration could also help to address the remaining problem

of handling missing stages that is still unsolved for our approach and other related reconstruction algorithms.

In summary, this thesis provides five contributions for both improved security monitoring and overall APT detection that help to lessen the burden on SOC analysts and threat hunters defending large networks. However, there are some areas that could benefit from additional improvements across the detection stack from event-level security monitoring up to detection of entire APT campaigns. This includes (i) a qualitative user study about selective TLS decryption, (ii) automated generation and maintenance of a security-aware asset database, (iii) generation of fine-grained system provenance datasets and evaluation of explainability approaches, (iv) application of AI-based approaches for detection of host-based APT stages, and (v) integration of CTI to our approach for APT contextualization based on KCSM.

A Appendix

A.1 Main Publications

[WF20b] Florian Wilkens and Mathias Fischer. "Towards Data-Driven Characterization of Brute-Force Attackers". In: *2020 IEEE Conference on Communications and Network Security (CNS)*. 2020 IEEE Conference on Communications and Network Security (CNS). Virtual Event (Avignon, France): IEEE, 2020-06, pages 1–9. ISBN: 978-1-72814-760-4. DOI: 10.1109/CNS48642.2020.9162326. (Visited on 02/10/2022).

[Wil+19a] Florian Wilkens, Steffen Haas, Dominik Kaaser, Peter Kling, and Mathias Fischer. "Towards Efficient Reconstruction of Attacker Lateral Movement". In: *Proceedings of the 14th International Conference on Availability, Reliability and Security*. ARES '19: 14th International Conference on Availability, Reliability and Security. Canterbury, United Kingdom: ACM, 2019-08-26, pages 1–9. ISBN: 978-1-4503-7164-3. DOI: 10.1145/3339252.3339254. (Visited on 02/10/2022).

[Wil+21] Florian Wilkens, Felix Ortmann, Steffen Haas, Matthias Vallentin, and Mathias Fischer. "Multi-Stage Attack Detection via Kill Chain State Machines". In: *Proceedings of the 3rd Workshop on Cyber-Security Arms Race*. CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security. Virtual Event (Seoul, Republic of Korea): ACM, 2021-11-19, pages 13–24. ISBN: 978-1-4503-8661-6. DOI: 10.1145/3474374.3486918. (Visited on 02/10/2022).

[Wil+22] Florian Wilkens, Steffen Haas, Johanna Amann, and Mathias Fischer. "Passive, Transparent, and Selective TLS Decryption for Network Security Monitoring". In: *ICT Systems Security and Privacy Protection*. IFIP SEC 2022. Edited by Weizhi Meng, Simone Fischer-Hübner, and Christian D. Jensen. Volume 648. IFIP Advances in Information and Communication Technology. Cham: Springer International Publishing, 2022-06-13, pages 87–105. ISBN: 978-3-031-06975-8. DOI: 10.1007/978-3-031-06975-8_6. (Visited on 01/09/2023).

[WWF23] Felix Welter, Florian Wilkens, and Mathias Fischer. "Tell Me More: Black Box Explainability for APT Detection on System Provenance Graphs". In: *Accepted for Publication at: 2023 IEEE International Conference on Communications (ICC): Communication and Information System Security Symposium (IEEE ICC'23 - CISS Symposium)*. Accepted for Publication at: 2023 IEEE International Conference on Communications (ICC): Communication and Information System Security Symposium (IEEE ICC'23 - CISS Symposium). Rome, Italy, 2023-05.

A.2 Additional Publications

[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)*. London, UK: IEEE, 2019-10, pages 1–11. DOI: 10.1109/IPCCC47392.2019.8958734.

[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*. Budapest, Hungary: IEEE, 2020-04, pages 1–6. DOI: 10.1109/NOMS47738.2020.9110470.

[Wil+19b] Florian Wilkens, Nurefsan Sertbas, Malte Hamann, and Mathias Fischer. "Towards Flexible Security Testing of OT Devices". In: *10. Jahreskolloquium Kommunikation in Der Automation (KommA 2019)*. Kommunikation in Der Automation (KommA 2019). Magdeburg, Germany, 2019-11, page 10.

A.3 Datasets

[WF20a] Florian Wilkens and Mathias Fischer. *Dataset: Brute-Force Logins 2020 (BFL2020)*. Dataset. 2020-04-20. DOI: 10.25592/uhhfdm.856.

[WHF22] Florian Wilkens, Steffen Haas, and Mathias Fischer. *Evaluation Dataset: Cooperative TLS-Decryption via Zeek*. Dataset. 2022-06-13. DOI: 10.25592/uhhfdm.10116.

A.4 Supervised Theses

[Grä19] Lars Leo Grätz. "Botmaster Attribution in Large-Scale P2P Botnets". Bachelor's thesis. Universität Hamburg, 2019-11-01.

[Kad20] Jan Kadel. "Detection of Advanced Persistent Threat Attacks via a Killchain-based Correlation of Host and Network Alerts". Bachelor's thesis. Universität Hamburg, 2020-12-11.

[Lau22] Jona Robin Laudan. "APT Detection for Enterprise Networks via Kill Chain State Machines with High Accuracy". Bachelor's thesis. Universität Hamburg, 2022-08-31.

[Lei20] Alexander Leib. "Dynamic Data-Driven Risk Assessment for Cloud Environments". Master's thesis. Universität Hamburg, 2020-07-20.

[Mar22] Finn Martens. "Aggregating Distributed Summary Statistics for Network Security Monitoring with Zeek". Master's thesis. Universität Hamburg, 2022-02-14.

[Poh20] Jan Pohlmann. "Designing a Stealthy, Distributed, Multi-Purpose Network and Malware Scanner". Bachelor's thesis. Universität Hamburg, 2020-10-28.

[Sch20] Henning Schütt. "Towards Transparent Decryption of TLS-encrypted End-User Traffic in Enterprise Networks". Master's thesis. Universität Hamburg, 2020-11-09.

[Sob19] Dennis Sobczak. "Summary Statistics in a Large-Scale Collaborative Intrusion Detection System Cluster". Master's thesis. Universität Hamburg, 2019-05-24.

[Wel22] Felix Welter. "Explainability in Machine Learning-based Intrusion Detection Systems". Master's thesis. Universität Hamburg, 2022-02-20.

Bibliography

[0xl22] 0xl3x1. *Zeek-EternalSafety*. 2022. URL: https://github.com/0xl3x1/zeek-EternalSafety (visited on 08/11/2022).

[40117] 401trg | Threat Research Group @ ProtectWise. *Detections/Pcaps at Master · 401trg/Detections*. GitHub. 2017-12-20. URL: https://github.com/401trg/detections (visited on 04/26/2022).

[AB02] Réka Albert and Albert-László Barabási. "Statistical Mechanics of Complex Networks". In: *Reviews of Modern Physics* 74.1 (2002-01-30), pages 47–97. ISSN: 0034-6861, 1539-0756. DOI: 10.1103/RevModPhys.74.47. (Visited on 02/10/2022).

[ABvO16] AbdelRahman Abdou, David Barrera, and Paul C. van Oorschot. "What Lies Beneath? Analyzing Automated SSH Bruteforce Attacks". In: *PASSWORDS 15: Technology and Practice of Passwords*. Edited by Frank Stajano, Stig F. Mjølsnes, Graeme Jenkinson, and Per Thorsheim. Volume 9551. Springer International Publishing, 2016, pages 72–91. ISBN: 978-3-319-29938-9. DOI: 10.1007/978-3-319-29938-9_6. (Visited on 02/10/2022).

[AL15] Michael J. Assante and Robert M. Lee. "The Industrial Control System Cyber Kill Chain". In: *SANS Institute Whitepapers* (2015-10-05). URL: https://www.sans.org/white-papers/36297/.

[Alg09] Algorithmics Group, University of Konstanz. *MDSJ: Java Library for Multidimensional Scaling*. 2009. URL: http://www.inf.uni-konstanz.de/algo/software/mdsj/.

[ALV08] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. "A Scalable, Commodity Data Center Network Architecture". In: *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication - SIGCOMM '08*. The ACM SIGCOMM 2008 Conference. Seattle, WA, USA: ACM, 2008-08, page 63. ISBN: 978-1-60558-175-0. DOI: 10.1145/1402958.1402967. (Visited on 02/10/2022).

[AM20] Almuthanna Alageel and Sergio Maffeis. *Hawk-Eye Dataset*. Dataset. 2020-12-17. DOI: 10.14469/hpc/7675. (Visited on 05/16/2022).

[AM21] Almuthanna Alageel and Sergio Maffeis. "Hawk-Eye: Holistic Detection of APT Command and Control Domains". In: *Proceedings of the 36th Annual ACM Symposium on Applied Computing*. SAC '21: The 36th ACM/SIGAPP Symposium on Applied Computing. Republic of Korea (Virtual Event): ACM, 2021-03-22, pages 1664–1673. ISBN: 978-1-4503-8104-8. DOI: 10.1145/3412841.3442040. (Visited on 02/28/2022).

[Ami+21] Md Ali Reza Al Amin, Sachin Shetty, Laurent Njilla, Deepak K. Tosh, and Charles Kamhoua. "Hidden Markov Model and Cyber Deception for the Prevention of Adversarial Lateral Movement". In: *IEEE Access* 9 (2021), pages 49662–49682. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3069105.

[AW22] Johanna Amann and Florian Wilkens. *Rudimentary Decryption for TLS 1.2 by 0xxon · Pull Request #1814 · Zeek/Zeek*. GitHub. 2022-03-02. URL: https://github.com/zeek/zeek/pull/1814 (visited on 10/31/2022).

[BBM15] Adam Bates, Kevin R. B. Butler, and Thomas Moyer. "Trustworthy Whole-System Provenance for the Linux Kernel". In: *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C., USA: USENIX Association, 2015-08, page 17. ISBN: 978-1-931971-23-2. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/bates.

[BC22] Kiran Bandla and Santiago Castro. *Aptnotes/Data*. Github: aptnotes/-data. 2022-09-26. URL: https://github.com/aptnotes/data (visited on 09/28/2022).

[Bej10] Robert Bejtlich. *What Is APT and What Does It Want?* TaoSecurity Blog. 2010-01-16. URL: https://web.archive.org/web/20220320044226/https://taosecurity.blogspot.com/2010/01/what-is-apt-and-what-does-it-want.html (visited on 04/05/2022).

[Bej20] Robert Bejtlich. *Greg Rattray Invented the Term Advanced Persistent Threat*. TaoSecurity Blog. 2020-10-10. URL: https://taosecurity.blogspot.com/2020/10/greg-rattray-invented-term-advanced.html (visited on 04/05/2022).

[Beu+17] Von Patrick Beuth, Kai Biermann, Martin Klingst, and Holger Stark. "Cyberattack on the Bundestag: Merkel and the Fancy Bear". In: *ZEIT ONLINE* (2017-05-12). URL: https://www.zeit.de/digital/2017-05/cyberattack-bundestag-angela-merkel-fancy-bear-hacker-russia (visited on 11/11/2022).

[Bie+17] Michael Bierma, Aaron Brown, Troy DeLano, Thomas M. Kroeger, and Howard Poston. "Locally Operated Cooperative Key Sharing (LOCKS)". In: *2017 International Conference on Computing, Networking and Communications (ICNC)*. 2017 International Conference on Computing, Networking and Communications (ICNC). Silicon Valley, CA, USA: IEEE, 2017-01, pages 356–362. ISBN: 978-1-5090-4588-4. DOI: 10.1109/ICCNC.2017.7876154. (Visited on 07/18/2022).

[Bor+17] Riccardo Bortolameotti et al. "DECANTeR: DEteCtion of Anomalous outbouNd HTTP TRaffic by Passive Application Fingerprinting". In: *Proceedings of the 33rd Annual Computer Security Applications Conference*. ACSAC 2017: 2017 Annual Computer Security Applications Conference. Orlando, FL, USA: ACM, 2017-12-04, pages 373–386. ISBN: 978-1-4503-5345-8. DOI: 10.1145/3134600.3134605. (Visited on 02/10/2022).

[BP04] Kevin Borders and Atul Prakash. "Web Tap: Detecting Covert Web Traffic". In: *Proceedings of the 11th ACM Conference on Computer and Communications Security - CCS '04*. The 11th ACM Conference. Washington DC, USA: ACM, 2004, page 110. ISBN: 978-1-58113-961-7. DOI: 10.1145/1030083.1030100. (Visited on 05/02/2022).

[Bro17] Kevin Broughton. *Automate Incident Response & Respond to Every Alert*. Swimlane [Security Operations]. 2017-03-03. URL: https://swimlane.com/blog/automated-incident-response-respond-every-alert (visited on 03/24/2022).

[Bro96] John Brooke. "SUS: A 'Quick and Dirty' Usability Scale". In: *Usability Evaluation In Industry*. Edited by Patrick W. Jordan, B. Thomas, Ian Lyall McClelland, and Bernard Weerdmeester. 1st edition. London, UK: CRC Press, 1996-06-11, pages 207–212. ISBN: 978-0-429-15701-1. DOI: 10.1201/9781498710411-35. (Visited on 10/30/2022).

[BS17] Blake D. Bryant and Hossein Saiedian. "A Novel Kill-Chain Framework for Remote Security Log Analysis with SIEM Software". In: *Computers*

& Security 67 (2017-06), pages 198–210. ISSN: 01674048. DOI: 10.1016/j.cose.2017.03.003. (Visited on 02/22/2022).

[BYG14] Parth Bhatt, Edgar Toshiro Yano, and Per Gustavsson. "Towards a Framework to Detect Multi-stage Advanced Persistent Threats Attacks". In: *2014 IEEE 8th International Symposium on Service Oriented System Engineering*. 2014 IEEE 8th International Symposium on Service Oriented System Engineering (SOSE). Oxford, United Kingdom: IEEE, 2014-04, pages 390–395. ISBN: 978-1-4799-3616-8. DOI: 10.1109/SOSE.2014.53. (Visited on 02/10/2022).

[Chi+11] Shih-Chuan Chiu, Hua-Fu Li, Jiun-Long Huang, and Hsin-Han You. "Incremental Mining of Closed Inter-Transaction Itemsets over Data Stream Sliding Windows". In: *Journal of Information Science* 37.2 (2011-04), pages 208–220. ISSN: 0165-5515, 1741-6485. DOI: 10.1177/0165551511401539. (Visited on 10/31/2022).

[CHK21] Aldo Cortesi, Maximilian Hils, and Thomas Kriechbaumer. *Mitmproxy — an Interactive TLS Proxy*. 2021. URL: https://mitmproxy.org.

[Cis18] Cision US Inc. *New Research from Advanced Threat Analytics Finds MSSP Incident Responders Overwhelmed by False-Positive Security Alerts*. 2018-02-12. URL: https://www.prnewswire.com/news-releases/new-research-from-advanced-threat-analytics-finds-mssp-incident-responders-overwhelmed-by-false-positive-security-alerts-300596828.html (visited on 03/24/2022).

[Cis22] Cisco. *Snort - Network Intrusion Detection & Prevention System*. Cisco, 2022. URL: https://www.snort.org/ (visited on 07/13/2022).

[CJ21] Katie Canales and Isabella Jibilian. *The US Is Readying Sanctions against Russia over the SolarWinds Cyber Attack. Here's a Simple Explanation of How the Massive Hack Happened and Why It's Such a Big Deal*. Business Insider. 2021-04-15. URL: https://www.businessinsider.com/solarwinds-hack-explained-government-agencies-cyber-security-2020-12 (visited on 11/11/2022).

[CNN16] CNN Editorial Research. "2016 Presidential Campaign Hacking Fast Facts". In: *CNN* (2016-12-27). URL: https://www.cnn.com/2016/12/26/us/2016-presidential-campaign-hacking-fast-facts/index.html (visited on 11/12/2022).

[Con+16] Andrea Continella et al. "ShieldFS: A Self-Healing, Ransomware-Aware Filesystem". In: *Proceedings of the 32nd Annual Conference on Computer Security Applications*. ACSAC '16. New York, NY, USA: ACM, 2016-12-05, pages 336–347. ISBN: 978-1-4503-4771-6. DOI: 10.1145/2991079.2991110. (Visited on 04/22/2022).

[Con17] Eric Conrad. *ShadowBrokers PCAPs, Etc.* Eric Conrad's blog | Author, SANS Faculty Fellow, and CTO of Backshore Communications. 2017-04-17. URL: https://www.ericconrad.com/2017/04/shadowbrokers-pcaps-etc.html (visited on 04/26/2022).

[Cor+04] L.P. Cordella, P. Foggia, C. Sansone, and M. Vento. "A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26.10 (2004-10), pages 1367–1372. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2004.75. (Visited on 02/10/2022).

[Cuc22] Cuckoo Contributors. *Cuckoo Sandbox — Automated Malware Analysis*. 2022. URL: https://cuckoosandbox.org/.

[DAR15] DARPA Information Innovation Office (I2O). *DARPA Transparent Computing Program*. 2015-12-04. URL: https://www.darpa.mil/program/transparent-computing (visited on 03/25/2022).

[DAR18] DARPA Information Innovation Office (I2O). *Transparent Computing Engagement 3 Data Release*. Dataset. 2018-08-30. URL: https://github.com/darpa-i2o/Transparent-Computing (visited on 03/25/2022).

[DAR20] DARPA Information Innovation Office (I2O). *Transparent Computing Engagement 5 Data Release*. Dataset. 2020-04-29. URL: https://github.com/darpa-i2o/Transparent-Computing (visited on 03/25/2022).

[dCM16] Xavier de Carné de Carnavalet and Mohammad Mannan. "Killed by Proxy: Analyzing Client-end TLS Interception Software". In: *Proceedings 2016 Network and Distributed System Security Symposium*. Network and Distributed System Security Symposium. San Diego, CA: Internet Society, 2016. ISBN: 978-1-891562-41-9. DOI: 10.14722/ndss.2016.23374. (Visited on 03/21/2022).

[Die17] Reinhard Diestel. *Graph Theory*. New York, NY: Springer Berlin Heidelberg, 2017. ISBN: 978-3-662-53621-6.

[Dij59] Edsger W. Dijkstra. "A Note on Two Problems in Connexion with Graphs". In: *Numerische mathematik* (1959).

[DPV05] Imre Derényi, Gergely Palla, and Tamás Vicsek. "Clique Percolation in Random Networks". In: *Physical Review Letters* 94.16 (2005-04-29), page 160202. ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.94.160202. (Visited on 03/16/2022).

[DRT09] Lorenzo De Nardo, Francesco Ranzato, and Francesco Tapparo. "The Subgraph Similarity Problem". In: *IEEE Transactions on Knowledge and Data Engineering* 21.5 (2009-05), pages 748–749. ISSN: 1558-2191. DOI: 10.1109/TKDE.2008.205.

[Dur+17] Zakir Durumeric et al. "The Security Impact of HTTPS Interception". In: *Proceedings 2017 Network and Distributed System Security Symposium*. Network and Distributed System Security Symposium. San Diego, CA: Internet Society, 2017. ISBN: 978-1-891562-46-4. DOI: 10.14722/ndss.2017.23456. (Visited on 03/21/2022).

[EKX96] Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". In: *KDD*. 1996, page 6.

[Ela22] Elasticsearch Contributors. *Elasticsearch*. Elasticsearch B.V., 2022. URL: https://github.com/elastic/elasticsearch (visited on 10/20/2022).

[Eng14] Giora Engel. *Deconstructing The Cyber Kill Chain*. Dark Reading. 2014-11-18. URL: https://www.darkreading.com/attacks-breaches/deconstructing-the-cyber-kill-chain (visited on 02/22/2022).

[FA16] Fatemeh Faraji Daneshgar and Maghsoud Abbaspour. "Extracting Fuzzy Attack Patterns Using an Online Fuzzy Adaptive Alert Correlation Framework: Fuzzy Adaptive Alert Correlation". In: *Security and Communication Networks* 9.14 (2016-09-25), pages 2245–2260. ISSN: 19390114. DOI: 10.1002/sec.1483. (Visited on 10/31/2022).

[Fir22] Firefox Contributors. *Firefox Browser*. Mozilla Corporation, 2022. URL: https://www.mozilla.org/en-US/firefox/new (visited on 11/01/2022).

[Fon+10] Romain Fontugne, Pierre Borgnat, Patrice Abry, and Kensuke Fukuda. "MAWILab : Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking". In: *Proceedings of the 6th International Conference on Emerging Networking EXperiments and*

Technologies. Co-NEXT '10: The 6th International Conference on Emerging Networking EXperiments and Technologies. Co-NEXT '10. Philadelphia, PA, USA: ACM, 2010-11-30, pages 1–12. ISBN: 978-1-4503-0448-1. DOI: 10.1145/1921168.1921179. (Visited on 04/21/2022).

[For15] Forum of Incident Response and Security Teams, Inc. "Common Vulnerability Scoring System Version 3.1". 2015. URL: https://www.first.org/cvss/v3-1/cvss-v31-specification_r1.pdf.

[Fra17] Julian Frangopoulos. "Automatische Generierung von Angriffspfaden zur Angriffserkennung und für digitale Forensik". Bachelor's thesis. Universität Hamburg, 2017-06-08.

[Fri+15] Ivo Friedberg, Florian Skopik, Giuseppe Settanni, and Roman Fiedler. "Combating Advanced Persistent Threats: From Network Event Correlation to Incident Detection". In: *Computers & Security* 48 (2015-02-01), pages 35–57. ISSN: 0167-4048. DOI: 10.1016/j.cose.2014.09.006. (Visited on 07/20/2022).

[Gep22] Gephi Consortium. *Gephi - The Open Graph Viz Platform*. Gephi, 2022. URL: https://github.com/gephi/gephi (visited on 09/04/2022).

[Goo22] Google Inc. *Google Transparency Report*. HTTPS encryption on the web – Google Transparency Report. 2022-08-16. URL: https://transparencyreport.google.com/https/overview (visited on 08/16/2022).

[Grä19] Lars Leo Grätz. "Botmaster Attribution in Large-Scale P2P Botnets". Bachelor's thesis. Universität Hamburg, 2019-11-01.

[Gre+09] Albert Greenberg et al. "VL2: A Scalable and Flexible Data Center Network". In: *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication - SIGCOMM '09*. The ACM SIGCOMM 2009 Conference. Barcelona, Spain: ACM, 2009-08, page 51. ISBN: 978-1-60558-594-9. DOI: 10.1145/1592568.1592576. (Visited on 02/10/2022).

[Gro22] Cisco Talos Intelligence Group. *PhishTank | Join the Fight against Phishing*. 2022-05. URL: https://www.phishtank.com/ (visited on 05/19/2022).

[GT12] Ashish Gehani and Dawood Tariq. "SPADE: Support for Provenance Auditing in Distributed Environments". In: *Middleware 2012*. Middleware 2012. Edited by Priya Narasimhan and Peter Triantafillou. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pages 101–120. ISBN: 978-3-642-35170-9. DOI: 10.1007/978-3-642-35170-9_6.

[Hah+15] Adam Hahn, Roshan K. Thomas, Ivan Lozano, and Alvaro Cardenas. "A Multi-Layered and Kill-Chain Based Security Analysis Framework for Cyber-Physical Systems". In: *International Journal of Critical Infrastructure Protection* 11 (2015-12), pages 39–50. ISSN: 18745482. DOI: 10.1016/j.ijcip.2015.08.003. (Visited on 02/11/2022).

[Han+20] Xueyuan Han, Thomas Pasquier, Adam Bates, James Mickens, and Margo Seltzer. "Unicorn: Runtime Provenance-Based Detector for Advanced Persistent Threats". In: *Proceedings 2020 Network and Distributed System Security Symposium*. Network and Distributed System Security Symposium. San Diego, CA, USA: Internet Society, 2020-02. ISBN: 978-1-891562-61-7. DOI: 10.14722/ndss.2020.24046. (Visited on 02/11/2022).

[Has+19] Wajih Ul Hassan et al. "NoDoze: Combatting Threat Alert Fatigue with Automated Provenance Triage". In: *Proceedings 2019 Network and Distributed System Security Symposium*. Network and Distributed System Security Symposium. San Diego, CA, USA: Internet Society, 2019-02-24. ISBN: 978-1-891562-55-6. DOI: 10.14722/ndss.2019.23349. (Visited on 02/11/2022).

[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: *Proceedings of the 6th International Conference on Information Warfare and Security*. 2011, page 14.

[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*. SAC 2018: Symposium on Applied Computing. Pau, France: ACM, 2018-04-09, pages 979–988. ISBN: 978-1-4503-5191-1. DOI: 10.1145/3167132.3167239. (Visited on 02/10/2022).

[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 (2019-04-08), pages 5–19. ISSN: 1559-6915. DOI: 10.1145/3325061.3325062. (Visited on 03/16/2022).

[HFS98] Steven A. Hofmeyr, Stephanie Forrest, and Anil Somayaji. "Intrusion Detection Using Sequences of System Calls". In: *Journal of Computer Security* 6.3 (1998-01-01), pages 151–180. ISSN: 0926-227X. DOI: 10.3233/JCS-980109. (Visited on 07/28/2022).

[Hil90] Mark D. Hill. "What Is Scalability?" In: *ACM SIGARCH Computer Architecture News* 18.4 (1990-12-02), pages 18–21. ISSN: 0163-5964. DOI: 10.1145/121973.121975. (Visited on 04/11/2022).

[HMW20] Junaid Haseeb, Masood Mansoori, and Ian Welch. "A Measurement Study of IoT-Based Attacks Using IoT Kill Chain". In: *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). IEEE, 2020-12, pages 557–567. DOI: 10.1109/TrustCom50675.2020.00080.

[HNR68] Peter Hart, Nils Nilsson, and Bertram Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pages 100–107. ISSN: 0536-1567. DOI: 10.1109/TSSC.1968.300136. (Visited on 02/10/2022).

[Hol13] Rick Holland. *Introducing Forrester's Cyber Threat Intelligence Research*. Forrester Blogs | Rick Holland's Blog. 2013-02-14. URL: https://web.archive.org/web/20140415054512/http://blogs.forrester.com/rick_holland/13-02-14-introducing_forresters_cyber_threat_intelligence_research (visited on 04/05/2022).

[Hon+22] Honeygrove Contributors et al. *Honeygrove: A Modular Python Honeypot Based on Broker and Twisted*. UHH/NET, 2022. URL: https://github.com/UHH-ISS/honeygrove (visited on 09/04/2022).

[Hos+17] Nahid Hossain et al. "SLEUTH: Real-time Attack Scenario Reconstruction from COTS Audit Data". In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC, Canada: USENIX Association, 2017-08-16, page 19. ISBN: 978-1-931971-40-9. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/hossain.

[Hos+20] Md Delwar Hossain, Hideya Ochiai, Fall Doudou, and Youki Kadobayashi. "SSH and FTP Brute-Force Attacks Detection in Computer Networks:

LSTM and Machine Learning Approaches". In: *2020 5th International Conference on Computer and Communication Systems (ICCCS)*. 2020 5th International Conference on Computer and Communication Systems (ICCCS). 2020-05, pages 491–497. DOI: 10.1109/ICCCS49078.2020.9118459.

[HS19] Cormac Herley and Stuart Schechter. "Distinguishing Attacks from Legitimate Authentication Traffic at Scale". In: *Proceedings 2019 Network and Distributed System Security Symposium*. Network and Distributed System Security Symposium. San Diego, CA, USA: Internet Society, 2019. ISBN: 978-1-891562-55-6. DOI: 10.14722/ndss.2019.23124. (Visited on 02/10/2022).

[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)*. London, UK: IEEE, 2019-10, pages 1–11. DOI: 10.1109/IPCCC47392.2019.8958734.

[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*. Budapest, Hungary: IEEE, 2020-04, pages 1–6. DOI: 10.1109/NOMS47738.2020.9110470.

[Hyn+20] Karel Hynek, Tomáš Beneš, Tomáš Čejka, and Hana Kubátová. "Refined Detection of SSH Brute-Force Attackers Using Machine Learning". In: *ICT Systems Security and Privacy Protection*. Edited by Marko Hölbl, Kai Rannenberg, and Tatjana Welzer. IFIP Advances in Information and Communication Technology. Cham: Springer International Publishing, 2020, pages 49–63. ISBN: 978-3-030-58201-2. DOI: 10.1007/978-3-030-58201-2_4.

[Jar12] Jeff Jarmoc. "SSL/TLS Interception Proxies and Transitive Trust". In: *Black Hat Europe 2012*. 2012, page 21.

[JP13] Mobin Javed and Vern Paxson. "Detecting Stealthy, Distributed SSH Brute-Forcing". In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*. CCS '13. New York, NY, USA: Association for Computing Machinery, 2013-11-04, pages 85–96. ISBN: 978-1-4503-2477-9. DOI: 10.1145/2508859.2516719. (Visited on 11/09/2022).

[KA22] Fred Klassen and AppNeta Contributors. *Tcpreplay*. AppNeta, Inc., 2022. URL: https://github.com/appneta/tcpreplay (visited on 11/01/2022).

[Kad20] Jan Kadel. "Detection of Advanced Persistent Threat Attacks via a Killchain-based Correlation of Host and Network Alerts". Bachelor's thesis. Universität Hamburg, 2020-12-11.

[Kam21] Hanife Kamen. "Honeypotbasierte Analyse von Angriffen auf Webanwendungen". Master's thesis. Universität Hamburg, 2021-10-10.

[KC03] Samuel T. King and Peter M. Chen. "Backtracking Intrusions". In: *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles - SOSP '03*. ACM Symposium on Operating Systems Principles. Bolton Landing, NY, USA: ACM, 2003, page 223. ISBN: 978-1-58113-757-6. DOI: 10.1145/945445.945467. (Visited on 02/11/2022).

[KC05] Samuel T. King and Peter M. Chen. "Backtracking Intrusions". In: *ACM Transactions on Computer Systems* 23.1 (2005-02-02), pages 51–76. ISSN: 0734-2071, 1557-7333. DOI: 10.1145/1047915.1047918. (Visited on 02/11/2022).

[KE10] Hugo Krawczyk and Pasi Eronen. *HMAC-based Extract-and-Expand Key Derivation Function (HKDF)*. Request for Comments (RFC) 5869. Internet Engineering Task Force (IETF), 2010-05. 14 pages. DOI: 10.17487/RFC5869. (Visited on 08/12/2022).

[KKK19] Hyeob Kim, HyukJun Kwon, and Kyung Kyu Kim. "Modified Cyber Kill Chain Model for Multimedia Service Environments". In: *Multimedia Tools and Applications* 78.3 (2019-02), pages 3153–3170. ISSN: 1380-7501, 1573-7721. DOI: 10.1007/s11042-018-5897-5. (Visited on 02/22/2022).

[Kli04] Ralf Klinkenberg. "Learning Drifting Concepts: Example Selection vs. Example Weighting". In: *Intelligent Data Analysis* 8.3 (2004-08-13), pages 281–300. ISSN: 15714128, 1088467X. DOI: 10.3233/IDA-2004-8305. (Visited on 03/18/2022).

[Lau22] Jona Robin Laudan. "APT Detection for Enterprise Networks via Kill Chain State Machines with High Accuracy". Bachelor's thesis. Universität Hamburg, 2022-08-31.

[Lee+19] Hyunwoo Lee et al. "maTLS: How to Make TLS Middlebox-Aware?" In: *Proceedings 2019 Network and Distributed System Security Symposium*. Network and Distributed System Security Symposium. San Diego, CA, USA: Internet Society, 2019. ISBN: 978-1-891562-55-6. DOI: 10.14722/ndss.2019.23547. (Visited on 02/16/2022).

[Lei20] Alexander Leib. "Dynamic Data-Driven Risk Assessment for Cloud Environments". Master's thesis. Universität Hamburg, 2020-07-20.

[Li+16] Meicong Li, Wei Huang, Yongbin Wang, Wenqing Fan, and Jianfang Li. "The Study of APT Attack Stage Model". In: *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*. IEEE/ACIS International Conference on Computer and Information Science (ICIS). Okayama, Japan: IEEE, 2016-06, pages 1–5. ISBN: 978-1-5090-0806-3. DOI: 10.1109/ICIS.2016.7550947. (Visited on 02/11/2022).

[Liu+18] Yushan Liu et al. "Towards a Timely Causality Analysis for Enterprise Security". In: *Proceedings 2018 Network and Distributed System Security Symposium*. Network and Distributed System Security Symposium. San Diego, CA, USA: Internet Society, 2018-02. ISBN: 978-1-891562-49-5. DOI: 10.14722/ndss.2018.23254. (Visited on 02/11/2022).

[LL17] Scott M Lundberg and Su-In Lee. "A Unified Approach to Interpreting Model Predictions". In: *Advances in Neural Information Processing Systems*. Volume 30. Long Beach, CA, USA: Curran Associates, Inc., 2017-12-04. URL: https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html (visited on 07/28/2022).

[Ln22] Gordon Lyon and nmap Contributors. *Nmap: The Network Mapper*. 2022. URL: https://nmap.org/ (visited on 07/26/2022).

[LTZ08] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation Forest". In: *2008 Eighth IEEE International Conference on Data Mining*. 2008 Eighth IEEE International Conference on Data Mining. 2008-12, pages 413–422. DOI: 10.1109/ICDM.2008.17.

[Ma+17] Shiqing Ma et al. "{MPI}: Multiple Perspective Attack Investigation with Semantic Aware Execution Partitioning". In: *26th USENIX Security Symposium (USENIX Security 17)*. 26th USENIX Security Symposium (USENIX Security 17). Vancouver, BC, Canada: USENIX Association, 2017-08-16, pages 1111–1128. ISBN: 978-1-931971-40-9. URL: https://www.

usenix.org/conference/usenixsecurity17/technical-sessions/presentation/ma (visited on 07/27/2022).

[Mal16] Sean T. Malone. "Using an Expanded Cyber Kill Chain Model to Increase Attack Resiliency". Black Hat USA 2016 (Las Vegas, NV, USA). 2016-08.

[Man10] Mandiant Inc. "Mandiant M-Trends: The Advanced Persistent Threat". In: *Mandiant M-Trends* (2010), page 32.

[Man13] Mandiant. *OpenIOC 1.1*. 2013. URL: https://github.com/fireeye/OpenIOC_1.1 (visited on 03/25/2022).

[Mar+16] Mirco Marchetti, Fabio Pierazzi, Michele Colajanni, and Alessandro Guido. "Analysis of High Volumes of Network Traffic for Advanced Persistent Threat Detection". In: *Computer Networks* 109 (2016-11), pages 127–141. ISSN: 13891286. DOI: 10.1016/j.comnet.2016.05.018. (Visited on 02/10/2022).

[Mar22] Finn Martens. "Aggregating Distributed Summary Statistics for Network Security Monitoring with Zeek". Master's thesis. Universität Hamburg, 2022-02-14.

[MC21] Wojciech Mazurczyk and Luca Caviglione. "Cyber Reconnaissance Techniques". In: *Communications of the ACM* 64.3 (2021-03), pages 86–95. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/3418293. (Visited on 04/22/2022).

[McD+10] Patrick McDaniel et al. "Towards a Secure and Efficient System for End-to-End Provenance". In: *Proceedings of the 2nd Conference on Theory and Practice of Provenance*. 2nd USENIX Workshop on the Theory and Practice of Provenance (TaPP '10). TAPP'10. San Jose, CA, USA: USENIX Association, 2010-02-22, page 2.

[MDN22] MDN Contributors. *Firefox Source Docs: NSS Key Log Format*. 2022. URL: https://firefox-source-docs.mozilla.org/security/nss/legacy/key_log_format/index.html (visited on 03/31/2022).

[Mei+13] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. "The TAMARIN Prover for the Symbolic Analysis of Security Protocols". In: *International Conference on Computer Aided Verification 2013*. CAV 2013: International Conference on Computer Aided Verification. Edited by Natasha Sharygina and Helmut Veith. Redacted by David Hutchison et al. Volume 8044. Computer Aided Verification. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pages 696–701. ISBN: 978-3-642-39798-1. DOI: 10.1007/978-3-642-39799-8_48. (Visited on 10/30/2022).

[MF21] Kathleen Moriarty and Stephen Farrell. *Deprecating TLS 1.0 and TLS 1.1*. Request for Comments (RFC) 8996. Internet Engineering Task Force (IETF), 2021-03. 18 pages. DOI: 10.17487/RFC8996. (Visited on 08/12/2022).

[Mic22] MicroFocus CyberRes. *ArcSight Enterprise Security Manager*. MicroFocus, 2022. URL: https://www.microfocus.com/en-us/cyberres/secops/arcsight-esm (visited on 07/27/2022).

[Mil+02] R. Milo et al. "Network Motifs: Simple Building Blocks of Complex Networks". In: *Science* 298.5594 (2002-10-25), pages 824–827. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.298.5594.824. (Visited on 07/26/2022).

[Mil+19a] Sadegh M. Milajerdi, Birhanu Eshete, Rigel Gjomemo, and V.N. Venkatakrishnan. "POIROT: Aligning Attack Behavior with Kernel Audit Records for Cyber Threat Hunting". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. New York, NY, USA: ACM, 2019-11-06, pages 1795–1812. ISBN: 978-1-4503-6747-9. DOI: 10.1145/3319535.3363217. (Visited on 03/18/2022).

[Mil+19b] Sadegh M. Milajerdi, Rigel Gjomemo, Birhanu Eshete, R. Sekar, and V.N. Venkatakrishnan. "HOLMES: Real-Time APT Detection through Correlation of Suspicious Information Flows". In: *2019 IEEE Symposium on Security and Privacy (SP)*. 2019 IEEE Symposium on Security and Privacy (SP). San Francisco, CA, USA: IEEE, 2019-05, pages 1137–1152. ISBN: 978-1-5386-6660-9. DOI: 10.1109/SP.2019.00026. (Visited on 02/11/2022).

[MIS22] MISP contributors. *MISP Open Source Threat Intelligence Platform & Open Standards For Threat Information Sharing*. 2022. URL: https://www.misp-project.org/ (visited on 03/25/2022).

[MIT00] MIT Lincoln Laboratory. *2000 DARPA Intrusion Detection Scenario Specific Datasets*. Dataset. 2000-07. URL: https://www.ll.mit.edu/r-d/datasets/2000-darpa-intrusion-detection-scenario-specific-datasets.

[MMA16] Emaad Manzoor, Sadegh M. Milajerdi, and Leman Akoglu. "Fast Memory-efficient Anomaly Detection in Streaming Heterogeneous Graphs". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. New York, NY, USA: ACM, 2016-08-13, pages 1035–1044. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939783. (Visited on 03/15/2022).

[Mül+01] Klaus-Robert Müller, Sebastian Mika, Gunnar Ratsch, Koji Tsuda, and Bernhard Schölkopf. "An Introduction to Kernel-Based Learning Algorithms". In: *IEEE Transactions on Neural Networks* 12.2 (2001-03), pages 181–201. ISSN: 10459227. DOI: 10.1109/72.914517. (Visited on 04/24/2022).

[Mun+06] Kiran-Kumar Muniswamy-Reddy, David A. Holland, Uri Braun, and Margo Seltzer. "Provenance-Aware Storage Systems". In: *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*. ATEC '06. Boston, MA, USA: USENIX Association, 2006-03-30, page 4.

[Naj+15] Maryam M. Najafabadi, Taghi M. Khoshgoftaar, Chad Calvert, and Clifford Kemp. "Detection of SSH Brute Force Attacks Using Aggregated Netflow Data". In: *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA). 2015-12, pages 283–288. DOI: 10.1109/ICMLA.2015.20.

[Nay+15] David Naylor et al. "Multi-Context TLS (mcTLS): Enabling Secure In-Network Functionality in TLS". In: *ACM SIGCOMM Computer Communication Review* 45.4 (2015-08-17), pages 199–212. ISSN: 0146-4833. DOI: 10.1145/2829988.2787482. (Visited on 02/16/2022).

[Nay+17] David Naylor, Richard Li, Christos Gkantsidis, Thomas Karagiannis, and Peter Steenkiste. "And Then There Were More: Secure Communication for More Than Two Parties". In: *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*. CoNEXT '17: The 13th International Conference on Emerging Networking EXperiments and Technologies. CoNext '17. Incheon, Republic of Korea: ACM, 2017-11-28, pages 88–100. ISBN: 978-1-4503-5422-6. DOI: 10.1145/3143361.3143383. (Visited on 02/16/2022).

[NDD21] Antonio José Horta Neto, Anderson Fernandes Pereira Dos Santos, and Marcos Dos Santos. "Polymer: An Adaptive Kill Chain Expanding Cyber Threat Hunting to Multi-Platform Environments". In: *2021 IEEE International Conference on Big Data (Big Data)*. 2021 IEEE International Conference on Big Data (Big Data). 2021-12, pages 2128–2135. DOI: 10.1109/BigData52589.2021.9671731.

[Net+22] NetworkX Developers, Aric Hagberg, Dan Schult, and Pieter Swart. *NetworkX*. NetworkX, 2022. URL: https://github.com/networkx/networkx (visited on 10/20/2022).

[NET21] NETRESEC AB. *PolarProxy TLS Proxy*. NETRESEC AB, 2021. URL: https://www.netresec.com/?page=PolarProxy.

[NSS22] NSS Contributors. *Network Security Services (NSS)*. Mozilla Corporation, 2022. URL: https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS (visited on 11/01/2022).

[OAS21] OASIS Cyber Threat Intelligence (CTI) TC. *Introduction to STIX*. 2021-06-10. URL: https://oasis-open.github.io/cti-documentation/stix/intro (visited on 03/25/2022).

[OGA05] Xinming Ou, Sudhakar Govindavajhala, and Andrew W. Appel. "MulVAL: A Logic-Based Network Security Analyzer". In: *14th USENIX Security Symposium (USENIX Security 05)*. SSYM'05. USA: USENIX Association, 2005-07-31, page 8.

[OM08] Jim Owens and Jeanna Matthews. "A Study of Passwords and Methods Used in Brute-Force SSH Attacks". In: *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*. 2008, page 8.

[Ope22] Open Information Security Foundation & Contributors. *Suricata*. 2022. URL: https://suricata.io/ (visited on 05/25/2022).

[Ort19] Felix Christian Ortmann. "Temporal and Spatial Alert Correlation for the Detection of Advanced Persistent Threats". Master's thesis. Universität Hamburg, 2019-07-18.

[OT22] Michel Oosterhof and Upi Tamminnen. *Cowrie SSH and Telnet Honeypot*. 2022. URL: https://www.cowrie.org/ (visited on 05/20/2022).

[Pas+17] Thomas Pasquier et al. "Practical Whole-System Provenance Capture". In: *Proceedings of the 2017 Symposium on Cloud Computing*. SoCC '17: ACM Symposium on Cloud Computing. Santa Clara, CA, USA: ACM, 2017-09-24, pages 405–418. ISBN: 978-1-4503-5028-0. DOI: 10.1145/3127479.3129249. (Visited on 02/11/2022).

[Pax99] Vern Paxson. "Bro: A System for Detecting Network Intruders in Real-Time". In: *7th USENIX Security Symposium (USENIX Security 99)*. Volume 7. San Antonio, TX, USA: USENIX Association, 1999-01. DOI: 10.1016/S1389-1286(99)00112-7.

[Ped+11] Fabian Pedregosa et al. "Scikit-Learn: Machine Learning in Python". In: *The Journal of Machine Learning Research (JMLR)* (2011), page 6.

[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*. ACSAC '16: 2016 Annual Computer Security Applications Conference. Los Angeles, CA, USA: ACM, 2016-12-05, pages 583–595. ISBN: 978-1-4503-4771-6. DOI: 10.1145/2991079.2991122. (Visited on 02/10/2022).

[Poh+12] Devin J. Pohly, Stephen McLaughlin, Patrick McDaniel, and Kevin Butler. "Hi-Fi: Collecting High-Fidelity Whole-System Provenance". In: *Proceedings of the 28th Annual Computer Security Applications Conference on - ACSAC '12*. The 28th Annual Computer Security Applications Conference. Orlando, Florida: ACM, 2012, page 259. ISBN: 978-1-4503-1312-4. DOI: 10.1145/2420950.2420989. (Visited on 02/11/2022).

[Poh20] Jan Pohlmann. "Designing a Stealthy, Distributed, Multi-Purpose Network and Malware Scanner". Bachelor's thesis. Universität Hamburg, 2020-10-28.

[Pol17] Paul Pols. "The Unified Kill Chain: Designing a Unified Kill Chain for Analyzing, Comparing and Defending against Cyber Attacks". Leiden, Netherlands: Cyber Security Academy (CSA), Leiden University, 2017-12-07. URL: https://www.unifiedkillchain.com/assets/The-Unified-Kill-Chain-Thesis.pdf.

[Pol21] Paul Pols. "The Unified Kill Chain - Raising Resilience against Advanced Cyber Attacks". 2021. URL: https://www.unifiedkillchain.com/assets/The-Unified-Kill-Chain.pdf.

[RAL17] Juan E. Rubio, Cristina Alcaraz, and Javier Lopez. "Preventing Advanced Persistent Threats in Complex Control Networks". In: *Computer Security*. ESORICS 2017. Edited by Simon N. Foley, Dieter Gollmann, and Einar Snekkenes. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pages 402–418. ISBN: 978-3-319-66399-9. DOI: 10.1007/978-3-319-66399-9_22.

[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*. Edited by Álvaro Herrero and Emilio Corchado. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2011, pages 58–67. ISBN: 978-3-642-21323-6. DOI: 10.1007/978-3-642-21323-6_8.

[RD08] Eric Rescorla and Tim Dierks. *The Transport Layer Security (TLS) Protocol Version 1.2*. Request for Comments (RFC) 5246. Internet Engineering Task Force (IETF), 2008-08. 104 pages. DOI: 10.17487/RFC5246. (Visited on 08/12/2022).

[Res18] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. Request for Comments (RFC) 8446. Internet Engineering Task Force (IETF), 2018-08. 160 pages. DOI: 10.17487/RFC8446. (Visited on 08/12/2022).

[RN03] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 2nd edition. Prentice Hall Series in Artificial Intelligence. Upper Saddle River, NJ, USA: Prentice Hall/Pearson Education, 2003. 1080 pages. ISBN: 978-0-13-790395-5.

[Ros11] Ronald S. Ross. "Managing Information Security Risk: Organization, Mission, and Information System View". In: *NIST Special Publication (NIST SP)* 800-39 (2011-03-01). URL: https://www.nist.gov/publications/managing-information-security-risk-organization-mission-and-information-system-view (visited on 04/05/2022).

[RSG16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. ""Why Should I Trust You?": Explaining the Predictions of Any Classifier". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016-08-13, pages 1135–1144. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939778. (Visited on 07/28/2022).

[Rub+18] Juan E. Rubio, Rodrigo Roman, Cristina Alcaraz, and Yan Zhang. "Tracking Advanced Persistent Threats in Critical Infrastructures Through Opinion Dynamics". In: *Computer Security*. ESORICS 2018. Edited by Javier Lopez, Jianying Zhou, and Miguel Soriano. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018-08-08, pages 555–574. ISBN: 978-3-319-99073-6. DOI: 10.1007/978-3-319-99073-6_27.

[SAN22] SANS Internet Storm Center. *DShield Data*. SANS Internet Storm Center. 2022-11-02. URL: http://www.dshield.org/data/index.html (visited on 11/02/2022).

[Sch20] Henning Schütt. "Towards Transparent Decryption of TLS-encrypted End-User Traffic in Enterprise Networks". Master's thesis. Universität Hamburg, 2020-11-09.

[Sea21] Jim Seaman. "Combating the Cyber-Security Kill Chain: Moving to a Proactive Security Model". In: *Artificial Intelligence in Cyber Security: Impact and Implications*. Edited by Reza Montasari and Hamid Jahankhani. Cham: Springer International Publishing, 2021, pages 121–155. ISBN: 978-3-030-88039-2. DOI: 10.1007/978-3-030-88040-8_5. (Visited on 11/11/2022).

[SEC22] SECEF. *IDMEF V1 | Overview*. 2022-05. URL: https://www.secef.net/secef/idmef/idmef-introduction/ (visited on 05/03/2022).

[Sec22] SecurityTrails. *SecurityTrails | Cyber Security API, Threat Intelligence API, Domain, DNS and IP Forensics*. 2022-05. URL: https://securitytrails.com/corp/api (visited on 05/16/2022).

[Sen22] SentinelOne. *What Is The Cyber Kill Chain?* SentinelOne. 2022-11-11. URL: https://www.sentinelone.com/cybersecurity-101/cyber-kill-chain/ (visited on 11/11/2022).

[SFF14] Florian Skopik, Ivo Friedberg, and Roman Fiedler. "Dealing with Advanced Persistent Threats in Smart Grid ICT Networks". In: *ISGT 2014*. ISGT 2014. Washington, DC, USA: IEEE, 2014-02, pages 1–5. DOI: 10.1109/ISGT.2014.6816388.

[Sha48] C. E. Shannon. "A Mathematical Theory of Communication". In: *Bell System Technical Journal* 27.3 (1948-07), pages 379–423. ISSN: 00058580. DOI: 10.1002/j.1538-7305.1948.tb01338.x. (Visited on 02/10/2022).

[SHG18] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization:" in: *Proceedings of the 4th International Conference on Information Systems Security and Privacy*. 4th International Conference on Information Systems Security and Privacy. Funchal, Madeira, Portugal: SCITEPRESS - Science and Technology Publications, 2018-01, pages 108–116. ISBN: 978-989-758-282-0. DOI: 10.5220/0006639801080116. (Visited on 02/10/2022).

[Shi+12] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A. Ghorbani. "Toward Developing a Systematic Approach to Generate Benchmark Datasets for Intrusion Detection". In: *Computers & Security* 31.3 (2012-05), pages 357–374. ISSN: 01674048. DOI: 10.1016/j.cose.2011.12.012. (Visited on 10/31/2022).

[Sko+14] Florian Skopik, Giuseppe Settanni, Roman Fiedler, and Ivo Friedberg. "Semi-Synthetic Data Set Generation for Security Software Evaluation". In: *2014 Twelfth Annual International Conference on Privacy, Security and Trust*. 2014 Twelfth Annual International Conference on Privacy, Security and Trust. Toronto, Canada: IEEE, 2014-07, pages 156–163. DOI: 10.1109/PST.2014.6890935.

[Sob19] Dennis Sobczak. "Summary Statistics in a Large-Scale Collaborative Intrusion Detection System Cluster". Master's thesis. Universität Hamburg, 2019-05-24.

[SR11] Guido Schwenk and Konrad Rieck. "Adaptive Detection of Covert Communication in HTTP Requests". In: *2011 Seventh European Conference on*

Computer Network Defense. 2011 Seventh European Conference on Computer Network Defense. Gothenburg, Sweden: IEEE, 2011-09, pages 25–32. DOI: 10.1109/EC2ND.2011.12.

[SS62] P. H. A. Sneath and Robert R. Sokal. "Numerical Taxonomy". In: *Nature* 193.4818 (4818 1962-03), pages 855–860. ISSN: 1476-4687. DOI: 10.1038/193855a0. (Visited on 04/21/2022).

[ST22] Salvadore Sanfilippo and The redis community. *Redis*. Redis, 2022. URL: https://github.com/redis/redis (visited on 09/04/2022).

[SV10] David Silver and Joel Veness. "Monte-Carlo Planning in Large POMDPs". In: *Advances in Neural Information Processing Systems*. Volume 23. Curran Associates, Inc., 2010. URL: https://proceedings.neurips.cc/paper/2010/hash/edfbe1afcf9246bb0d40eb4d8027d90f-Abstract.html (visited on 07/27/2022).

[Tcp22] Tcpdump Group. *Tcpdump*. The Tcpdump Group, 2022. URL: https://github.com/the-tcpdump-group/tcpdump (visited on 07/25/2022).

[Ten22] Tenable Inc. *Nessus® Vulnerability Assessment*. 2022. URL: https://www.tenable.com/products/nessus (visited on 07/26/2022).

[The22a] The MITRE Corporation. *BZAR (Bro/Zeek ATT&CK-based Analytics and Reporting)*. MITRE ATT&CK, 2022. URL: https://github.com/mitre-attack/bzar (visited on 08/11/2022).

[The22b] The MITRE Corporation. *MITRE ATT&CK®*. MITRE ATT&CK®. 2022-08-11. URL: https://attack.mitre.org/ (visited on 08/11/2022).

[Tia+19] Zhihong Tian et al. "Real-Time Lateral Movement Detection Based on Evidence Reasoning Network for Edge Computing Environment". In: *IEEE Transactions on Industrial Informatics* 15.7 (2019-07), pages 4285–4294. ISSN: 1941-0050. DOI: 10.1109/TII.2019.2907754.

[TYH22] The OpenSSL Project, Eric A. Young, and Tim J. Hudson. *OpenSSL - TLS/SSL and Crypto Library*. OpenSSL, 2022. URL: https://github.com/openssl/openssl (visited on 10/29/2022).

[UC 04] UC Santa Barbara. *Treasure Hunt Dataset*. Dataset. 2004. URL: http://www.cs.ucsb.edu/rsg/datasets%20%3Cno%20longer%20available%20online%3E.

[Uni17] University of Twente. *Data Exfiltration Malware (DEM) | Services and Cyber-Security Group | University of Twente*. Universiteit Twente. 2017. URL: https://www.utwente.nl/en/eemcs/scs/output/downloads/20171127_DEM/ (visited on 04/26/2022).

[Uss+16] Martin Ussath, David Jaeger, Feng Cheng, and Christoph Meinel. "Advanced Persistent Threats: Behind the Scenes". In: *2016 Annual Conference on Information Science and Systems (CISS)*. 2016 Annual Conference on Information Science and Systems (CISS). Princeton, NJ, USA: IEEE, 2016-03, pages 181–186. ISBN: 978-1-4673-9457-4. DOI: 10.1109/CISS.2016.7460498. (Visited on 02/10/2022).

[Vas+13] Emmanouil Vasilomanolakis et al. "This Network Is Infected: HosTaGe - a Low-Interaction Honeypot for Mobile Devices". In: *Proceedings of the Third ACM Workshop on Security and Privacy in Smartphones & Mobile Devices - SPSM '13*. Berlin, Germany: ACM Press, 2013, pages 43–48. ISBN: 978-1-4503-2491-5. DOI: 10.1145/2516760.2516763. (Visited on 02/10/2022).

[Wat16] Dave Watson. "KTLS: Linux Kernel Transport Layer Security". In: *Netdev 1.2*. Tokyo, Japan, 2016, page 4. URL: https://netdevconf.org/1.2/papers/ktls.pdf (visited on 02/10/2022).

[Wel22] Felix Welter. "Explainability in Machine Learning-based Intrusion Detection Systems". Master's thesis. Universität Hamburg, 2022-02-20.

[WF20a] Florian Wilkens and Mathias Fischer. *Dataset: Brute-Force Logins 2020 (BFL2020)*. Dataset. 2020-04-20. DOI: 10.25592/uhhfdm.856.

[WF20b] Florian Wilkens and Mathias Fischer. "Towards Data-Driven Characterization of Brute-Force Attackers". In: *2020 IEEE Conference on Communications and Network Security (CNS)*. 2020 IEEE Conference on Communications and Network Security (CNS). Virtual Event (Avignon, France): IEEE, 2020-06, pages 1–9. ISBN: 978-1-72814-760-4. DOI: 10.1109/CNS48642.2020.9162326. (Visited on 02/10/2022).

[WHF22] Florian Wilkens, Steffen Haas, and Mathias Fischer. *Evaluation Dataset: Cooperative TLS-Decryption via Zeek*. Dataset. 2022-06-13. DOI: 10.25592/uhhfdm.10116.

[Wil+19a] Florian Wilkens, Steffen Haas, Dominik Kaaser, Peter Kling, and Mathias Fischer. "Towards Efficient Reconstruction of Attacker Lateral Movement". In: *Proceedings of the 14th International Conference on Availability, Reliability and Security*. ARES '19: 14th International Conference on Availability, Reliability and Security. Canterbury, United Kingdom: ACM, 2019-08-26, pages 1–9. ISBN: 978-1-4503-7164-3. DOI: 10.1145/3339252.3339254. (Visited on 02/10/2022).

[Wil+19b] Florian Wilkens, Nurefsan Sertbas, Malte Hamann, and Mathias Fischer. "Towards Flexible Security Testing of OT Devices". In: *10. Jahreskolloquium Kommunikation in Der Automation (KommA 2019)*. Kommunikation in Der Automation (KommA 2019). Magdeburg, Germany, 2019-11, page 10.

[Wil+21] Florian Wilkens, Felix Ortmann, Steffen Haas, Matthias Vallentin, and Mathias Fischer. "Multi-Stage Attack Detection via Kill Chain State Machines". In: *Proceedings of the 3rd Workshop on Cyber-Security Arms Race*. CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security. Virtual Event (Seoul, Republic of Korea): ACM, 2021-11-19, pages 13–24. ISBN: 978-1-4503-8661-6. DOI: 10.1145/3474374.3486918. (Visited on 02/10/2022).

[Wil+22] Florian Wilkens, Steffen Haas, Johanna Amann, and Mathias Fischer. "Passive, Transparent, and Selective TLS Decryption for Network Security Monitoring". In: *ICT Systems Security and Privacy Protection*. IFIP SEC 2022. Edited by Weizhi Meng, Simone Fischer-Hübner, and Christian D. Jensen. Volume 648. IFIP Advances in Information and Communication Technology. Cham: Springer International Publishing, 2022-06-13, pages 87–105. ISBN: 978-3-031-06975-8. DOI: 10.1007/978-3-031-06975-8_6. (Visited on 01/09/2023).

[Wil92] Frank Wilcoxon. "Individual Comparisons by Ranking Methods". In: *Breakthroughs in Statistics: Methodology and Distribution*. Edited by Samuel Kotz and Norman L. Johnson. Springer Series in Statistics. New York, NY: Springer, 1992, pages 196–202. ISBN: 978-1-4612-4380-9. DOI: 10.1007/978-1-4612-4380-9_16. (Visited on 10/16/2022).

[Wir22] Wireshark Foundation. *Wireshark · GitLab*. Wireshark Foundation, 2022. URL: https://gitlab.com/wireshark/wireshark (visited on 11/01/2022).

[WOL22] Florian Wilkens, Felix Christian Ortmann, and Jona Laudan. *APT Contextualizer*. Version v2. UHH/NET, 2022. URL: https://github.com/UHH-ISS/apt-contextualizer.

[WWF23] Felix Welter, Florian Wilkens, and Mathias Fischer. "Tell Me More: Black Box Explainability for APT Detection on System Provenance Graphs". In:

Accepted for Publication at: 2023 IEEE International Conference on Communications (ICC): Communication and Information System Security Symposium (IEEE ICC'23 - CISS Symposium). Accepted for Publication at: 2023 IEEE International Conference on Communications (ICC): Communication and Information System Security Symposium (IEEE ICC'23 - CISS Symposium). Rome, Italy, 2023-05.

[Xie+13] Yulai Xie, Kiran-Kumar Muniswamy-Reddy, Dan Feng, Yan Li, and Darrell D. E. Long. "Evaluation of a Hybrid Approach for Efficient Provenance Storage". In: *ACM Transactions on Storage* 9.4 (2013-11-01), 14:1–14:29. ISSN: 1553-3077. DOI: 10.1145/2501986. (Visited on 07/27/2022).

[Xie+16] Yulai Xie, Dan Feng, Zhipeng Tan, and Junzhe Zhou. "Unifying Intrusion Detection and Forensic Analysis via Provenance Awareness". In: *Future Generation Computer Systems* 61 (2016-08), pages 26–36. ISSN: 0167739X. DOI: 10.1016/j.future.2016.02.005. (Visited on 04/05/2022).

[Xie+20] Yulai Xie et al. "Pagoda: A Hybrid Approach to Enable Efficient Real-Time Provenance Based Intrusion Detection in Big Data Environments". In: *IEEE Transactions on Dependable and Secure Computing* 17.6 (2020-11), pages 1283–1296. ISSN: 1941-0018. DOI: 10.1109/TDSC.2018.2867595.

[Yam+15] Masahiro Yamada, Masanobu Morinaga, Yuki Unno, Satoru Torii, and Masahiko Takenaka. "RAT-based Malicious Activities Detection on Enterprise Internal Networks". In: *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*. 2015 10th International Conference for Internet Technology and Secured Transactions (ICITST). London, UK: IEEE, 2015-12, pages 321–325. ISBN: 978-1-908320-52-0. DOI: 10.1109/ICITST.2015.7412113. (Visited on 10/25/2022).

[Yan+17] Dingqi Yang, Bin Li, Laura Rettig, and Philippe Cudré-Mauroux. "HistoSketch: Fast Similarity-Preserving Sketching of Streaming Histograms with Concept Drift". In: *2017 IEEE International Conference on Data Mining (ICDM)*. 2017 IEEE International Conference on Data Mining (ICDM). 2017-11, pages 545–554. DOI: 10.1109/ICDM.2017.64.

[Yen71] Jin Y. Yen. "Finding the K Shortest Loopless Paths in a Network". In: *Management Science* 17.11 (1971-07), pages 712–716. ISSN: 0025-1909, 1526-5501. DOI: 10.1287/mnsc.17.11.712. (Visited on 02/10/2022).

[Zan+03] Tom Zanussi, Karim Yaghmour, Robert Wisniewski, Richard Moore, and Michel Dagenais. "Relayfs: An Efficient Unified Approach for Transmitting Data from Kernel to User Space". In: *Proceedings of the Ottawa Linux Symposium 2003*. Ottawa, Canada, 2003-07-23.

[Zee22] Zeek Project Contributors. *The Zeek Network Security Monitor*. 2022. URL: https://zeek.org.

[Zet14] Kim Zetter. "An Unprecedented Look at Stuxnet, the World's First Digital Weapon". In: *Wired* (2014-11-03). ISSN: 1059-1028. URL: https://www.wired.com/2014/11/countdown-to-zero-day-stuxnet/ (visited on 11/11/2022).

List of Figures

List of Tables

List of Algorithms

List of Abbreviations & Acronyms

ACM Association for Computing Machinery

AEAD authenticated encryption with associated data

AES Advanced Encryption Standard

AG attack graph

AI artificial intelligence

API application programming interface

APT advanced persistent threat

AS autonomous system

AUB area under baseline

BAG Bayesian attack graph

BFS breadth-first search

BNF Backus-Naur form

BSD Berkeley Software Distribution

BZAR Bro/Zeek ATT&CK-based Analytics and Reporting

C2/C&C command & control

CDF cumulative distribution function

CDN content delivery network

CPM clique percolation method

CPS cyber physical system

CPU central processing unit

CSG communication structure graph

CTI cyber threat intelligence

CV certificate validation

CVE Common Vulnerabilities and Exposures

CVSS Common Vulnerability Scoring System

DARPA Defense Advanced Research Projects Agency

DBSCAN density-based spatial clustering of applications with noise

DDoS distributed denial of service

DECANTeR DEteCtion of Anomalous outbouNd HTTP TRaffic by Passive Application Fingerprinting

DFS depth-first search

DMZ demilitarized zone

DNS Domain Name System

DoS denial of service

DSA Digital Signature Algorithm

DSL domain-specific language

DtA distance to asset

ECDHE Elliptic-curve Diffie–Hellman Ephemeral

ENKCSM Extended Network Kill Chain State Machine

ERN evidence reasoning network

ESM enterprise security manager

FPR false positive rate

FQDN fully-qualified domain name

FTP File Transfer Protocol

GAC Graph-based Alert Correlation

GCM Galois/Counter Mode

GPL GNU General Public License

GPM graph pattern matching

HDFS Hadoop Distributed File System

HIDS host intrusion detection system

HKDF HMAC-based Extract-and-Expand Key Derivation Function

HMAC hash-based message authentication code

HMM hidden Markov model

HSG High-level Scenario Graph

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

ICS industrial control system

ICSI International Computer Science Institute

IDMEF Intrusion Detection Message Exchange Format

IDS intrusion detection system

IEEE Institute of Electrical and Electronics Engineers

IFIP International Federation for Information Processing

IKC intrusion kill chain

IoC indicator of compromise

IoT Internet of Things

IoTKC Internet of Things Kill Chain

IP Internet Protocol

IPS intrusion prevention system

ISP Internet Service Provider

IT Information Technology

IV initialization vector

JSON JavaScript Object Notation

KCSM Kill Chain State Machine

KE key exchange

kTLS in-kernel Transport Layer Security

LIME Local Interpretable Model-agnostic Explanations

LM lateral movement

LoC lines of code

LOCKS Locally Operated Cooperative Key Sharing

LPM Linux Provenance Modules

LSM Linux Security Modules

maTLS Middlebox-aware TLS

mbTLS Middlebox TLS

mcTLS Multi-Context TLS

MitM Man-in-the-Middle

ML machine learning

NIST US National Institute of Standards and Technology

NKCSM Network Kill Chain State Machine

NMS network monitoring system

NSS Network Security Services (library)

OCSP Online Certificate Status Protocol

OISF Open Information Security Foundation

OS operating system

OSINT open-source intelligence

OT Operational Technology

PaaS platform as a service

PASS Provenance-Aware Storage System

PDF Portable Document Format

PIDAS Provenance-aware Intrusion Detection and Analysis System

PKI public-key infrastructure

PLA People's Liberation Army

PoC proof-of-concept

POMCP partially observable Monte-Carlo planning

PRF pseudo-random function

PSK pre-shared key

QA quality assurance

QoS Quality of Service

R&D Research and Development

RAM Random Access Memory

RAT remote access trojan/remote administration tool

RCE remote code execution

rDNS reverse DNS

RDP remote desktop protocol

RFC Request for Comments

ROC receiver operating characteristic

RSA Rivest-Shamir-Adleman

SCADA supervisory control and data acquisition

SDK software development kit

SGX Software Guard Extensions

SHAP SHapley Additive exPlanations

SIEM security information and event management

SMB Server Message Block

SMTP Simple Mail Transfer Protocol

SOC security operations center

SSH Secure Shell

SSL Secure Sockets Layer

SUS System Usability Scale

SVM support vector machine

TbL Time-between-Logins

TbS Time-between-Sessions

TCB trusted computing base

TCP Transport Control Protocol

TLD top-level domain

TLS Transport Layer Security

TPM trusted platform module

TPR true positive rate

TTP tactics, techniques, and procedures

UDP User Datagram Protocol

UKC unified kill chain

UP-GMA unweighted pair group method with arithmetic mean

URI Uniform Resource Identifier

US United States

USB Universal Serial Bus

VM virtual machine

VPN virtual private network

XAI explainable artificial intelligence

XML Extensible Markup Language

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Dissertationsschrift mit dem Titel „Methods for Enhanced Security Monitoring and APT Detection in Enterprise Networks“ selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich bin weiterhin damit einverstanden, dass meine Dissertationsschrift in den Bestand der Fachbereichsbibliothek eingestellt wird.

Hamburg, Deutschland, 15.11.2022

.....

Florian Wilkens