# Cross-Machine Multi-Phase Advanced Persistent Threat Detection and Investigation via Provenance Analytics

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik des
Karlsruher Instituts für Technologie (KIT)

genehmigte
Dissertation

von

M.Sc. Qi Liu

Tag der mündlichen Prüfung: 07. Februar 2025
1. Referent: Prof. Dr. Veit Hagenmeyer
2. Referent: Prof. Dr. Mathias Fischer

# Abstract

Attack detection and investigation are an iterative process in practice, in which security analysts still play an important role as of today. Security systems for attack detection and investigation need to be designed with this human-in-the-loop aspect in mind. A practical, reliable attack detection system is not just a classification system. Rather, it facilitates the investigation process in unearthing the root causes and attack ramifications, by providing contextualized and more interpretable detection results. Security analysts often find it difficult and time consuming to investigate on, associate and understand the detection results of currently deployed security systems. A swift and accurate attack detection & investigation process is crucial for timely and proper attack recovery & remediation.

To support speedy and thorough attack detection & investigation, provenance-based security systems have been proposed over the past few years. These systems have proven to be inherently suitable for this critical mission: providing security analysts with insightful, contextualized, and actionable detection results for further investigation in a highly automated manner. Provenance-based systems produce attack graphs by parsing system logs recording what has occurred in a computer system at a fine-granular level. Such graphs manifest and link causally related system activities. Given a suspicious event as a starting point, a backward tracing and forward tracing in a graph can quickly expose more related malicious system activities caused by attackers, i.e., the root cause and attack ramifications, respectively. Provenance-based security systems have demonstrated excellent performance in reducing false alarms, supplying security analysts with accurate and self-explanatory attack graphs, in particular for sophisticated attacks conducted by Advanced Persistent Threat (APT) actors.

Despite the success, our examination of existing provenance-based systems yields the finding that these systems suffer from several major limitations, and can be rendered ineffective facing evasive real-world APT actors. First,

they are fundamentally susceptible to evasive attacks employing persistence techniques. Second, existing provenance-based systems are limited to tracing inside a single machine, and unable to trace across machines and reveal the extend of attackers' traversal inside a network. Third, these systems only process system logs from general-purpose OS like Windows and Linux, but not logs from devices of embedded systems.

To tackle the first weakness of prior provenance-based systems, we present CPD, which is, to our knowledge, the first system specialized for persistence detection, and hence multi-phase APT detection and investigation. CPD is powered by two novel concepts: *pseudo-dependency edges*, which effectively reconnect fragmented attack graphs resulted from persistence techniques, and *expert-guided edges*, which capacitate faster tracing and reduced log size. Moreover, we create HADES to overcome the second constraint of provenance-based systems in combating APT actors. HADES demonstrates, to our knowledge, the first approach capable of performing accurate causality-based cross-machine tracing. In HADES, we introduce a lightweight authentication anomaly detection model and a novel concept called *logon session-based execution partitioning and tracing*, which together empower efficient and accurate cross-machine APT detection and investigation.

Last, we design our third system COMMANDER to further strengthen the defense line against cross-machine multi-phase APT attacks. Our extensive analysis of APT threat reports reveals that HADES's cross-machine tracing functionality is vulnerable to several evasive attack techniques routinely employed by APT actors: persistence, session hijacking, and port forwarding. Recognizing this, we introduce a modular design in COMMANDER, in which it integrates CPD with HADES, and incorporates another two specialized detectors for session hijacking and port forwarding, respectively. These specialized detectors make complementary contributions to safeguarding robust and correct whole network tracing, by delivering critical information to guide and adjust the tracing process. Moreover, COMMANDER is designed for detection and investigation of attacks against industrial-sector organizations. That is, it includes parsers for logs from some popular industrial controllers, and detection rules for attack techniques of APT actors on industrial control systems. COMMANDER can accurately attribute the malicious system activities on these industrial controllers to the true identity behind those actions, even if the access originates from the enterprise networks.

# Acknowledgment

In my journey to pursue the PhD degree, I encountered several big obstacles that cast into doubt and some discouraging moments. Without the support of a number of people, this dissertation would not be possible.

First, I am very grateful to my supervisor Prof. Veit Hagenmeyer and my group leader Dr. Kaibin Bao for their valuable research advice for me, trust on me, and being patient with me during my entire PhD journey. Besides, I thank them for giving me the opportunity, freedom, encouragement and support to research on what really interests me.

Second, I want to thank all colleagues who have provided me help at our institute IAI. I seldom take things for granted, and always try to remember every support I received. I am not going to list you all here. But if you are an IAI colleague reading this, yes, I mean you. Saying so does not mean I want to express my appreciation only to colleagues in my research area. I am equally tankful to all administrative staff, secretarial staff and IT staff at IAI. Looking back on the past years, I have been very happy with my working environment at IAI, in which people are mindful and respectful to each other.

Third, I want to thank Prof. Wajih Ul Hassen from University of Virginia for providing me the opportunity to have a four-month research stay at his DART lab, and guiding me with his valuable research advice. I also want to thank the Karlsruhe House of Young Scientists (KHYS) for the financial support during my research stay at University of Virginia.

Fourth, I want to thank Prof. Mathias Fischer from University of Hamburg for agreeing to be the second referee of my PhD thesis, inviting me over to present my PhD topic, and giving me valuable feedback. I also thank Prof. Jürgen Beyerer at KIT for inviting me over to discuss my PhD topic, and agreeing to join the committee for my PhD exam.

Last, I am deeply grateful to my parents for teaching me to become a more responsible, caring, humble and diligent person. I cannot image a more responsible father and a more caring mother.

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# Acronyms

| | |
|---|---|
| **AD** | Active Discovery |
| **API** | Application Programming Interface |
| **APT** | Advanced Persistent Threat |
| **ALPC** | Asynchronous Local Inter-Process Communication |
| **AS** | Authentication Service |
| **C2** | Command and Control |
| **CBC** | Carbon Black Cloud |
| **CDF** | Cumulative Distribution Function |
| **CKC** | Cyber Kill Chain |
| **DC** | Domain Controller |
| **DLL** | Dynamic-Link Library |
| **DNS** | Domain Name System |
| **EDR** | Endpoint Detection and Response |
| **EQL** | Event Query Language |
| **ETW** | Event Tracing for Windows |
| **FN** | False Negative |
| **FP** | False Positive |
| **FQDN** | Fully Qualified Domain Name |
| **FSA** | Finite State Automata |
| **GNN** | Graph Neural Networks |

| | |
|---|---|
| **GUID** | Globally Unique ID |
| **HSG** | High-level Scenario Graph |
| **IAM** | Identity and Access Management |
| **ICS** | Industrial Control System |
| **IDS** | Intrusion Detection Systems |
| **IED** | Intelligent Electronic Device |
| **IOC** | Indicator Of Compromise |
| **IPC** | Inter Process Communication |
| **LAN** | Local Area Network |
| **LDAP** | Lightweight Directory Access Protocol |
| **LOLBins** | Living Off the Land Binaries |
| **ML** | Machine Learning |
| **NIDS** | Network-based Intrusion Detection System |
| **NTLM** | New Technology LAN Manager |
| **OS** | Operating System |
| **OpTC** | Operationally Transparent Cyber |
| **OT** | Operational Technology |
| **PIDS** | Provenance-based Intrusion Detection System |
| **PLC** | Programmable Logic Controller |
| **RDP** | Remote Desktop Protocol |
| **RTU** | Remote Terminal Unit |
| **SIEM** | Security Information and Event Management |
| **SMB** | Server Message Block |
| **SOC** | Security Operation Center |
| **SPN** | Service Principal Name |
| **TC** | Transparent Computing |

| | |
|---|---|
| **TCB** | Trusted Computing Base |
| **TGS** | Ticket-Granting Service |
| **TGT** | Ticket-Granting-Ticket |
| **TTP** | Tactics, Techniques, Procedures |
| **WMI** | Windows Management Instrumentation |
| **XDR** | eXtended Detection and Response |

# 1 Introduction

Advanced Persistent Threat (APT) actors are well-resourced attackers who specialize in launching covert cyberattacks on computer systems to which they maintain unauthorized access for an extended period of time [1], [2]. To describe the systematic attack process of APT attacks, the term Cyber Kill Chain [3] was coined. The main objective of APT actors is cyber-espionage [4]–[8] or impairing physical processes at a *strategic* point when attacking critical infrastructures [9]–[13], rather than causing immediate destruction to the target organizations after the initial access. Once inside the target systems, APT actors typically monitor the systems to find and exfiltrate sensitive data. To remain undetected for as long as possible, these attackers routinely employ the so-called low-and-slow strategy [14].

To stay low, APT actors often avoid using self-developed or third-party hacking tools, and instead rely on Living Off the Land Binaries (LOLBins) [15]–[17], i.e., system pre-installed programs, which are used by system administrators and genuine users as well. Leveraging these programs helps to evade detection, as the associated system activities are often considered as benign by detection systems due to their constant presence during normal operations. Conventional intrusion detection systems (IDS) commonly focus on isolated system events, and excel at identifying malware causing a burst of malicious behaviors. Facing APT actors frequently employing LOLBins, these IDS tend to miss those attacks and suffer from a high false negative rate [18], [19].

To overcome this, security vendors resort to the MITRE ATT&CK Matrix [20] as a reference for further creating detection rules. The MITRE ATT&CK Matrix, maintained with contributions from leading industrial security vendors, provides a high-level summary of attack tactics and techniques based on real-world observations, including usage of LOLBins. Simply applying the detection rules derived from the MITRE ATT&CK Matrix, however, results in a deluge of security alerts, and hence leads to the alert fatigue problem. In fact, the majority of these alerts are false positives [21]. Our evaluations

confirm that the isolated analysis of these IDS create an excessive amount of false alerts due to the fact that LOLBins are programs executed frequently also by genuine users.

Provenance-based intrusion detection systems (PIDS) [22]–[36] emerged as a promising new solution, associating causally related system activities by parsing system events like process creation, file access and network socket access into provenance graphs. Given a suspicious event as a starting point, a backward tracing and forward tracing in the provenance graph can expose more related malicious system activities caused by attackers, i.e., the root cause and attack ramifications, respectively. PIDS have proven to be effective in particular in reducing false alarm rates and presenting real attack activities in attack graphs. The underlying assumption of most PIDS [22], [24]–[27], [29], [31], [34] is that, when several alerts can be causally linked into an attack graph reassembling a Cyber Kill Chain, they are much more likely caused by attackers than those alerts unrelated to each other. Besides, attack graphs provide much richer context for security analysts to further investigate the scope of an attack, invaluable for an effective and efficient security operation center (SOC) in practice.

*However, we find that existing PIDS suffer from two major limitations, significantly lowering their effectiveness in detecting real-world APT attacks.* First, they are fundamentally vulnerable to evasive attacks employing persistence techniques. While utilizing LOLBins makes APT actors stay low, to stay slow, they typically rely on persistence techniques, granting them prolonged access to victim systems even after unforeseen interruptions like machine restarts, credential changes or remote connection losses. Our investigation reveals that about 70% of real-world APT groups leveraged at least one persistence technique in the past (Chapter 4). Leveraging persistence techniques can fragment an attack graph back to seemingly unrelated security alerts in multiple phases, rendering existing PIDS ineffective.

Second, existing PIDS are restricted to tracing inside a single machine, and unable to reveal the scope of attackers' traversal inside a network, crucial for accurate attack detection and investigation in enterprise networks. In real-world APT attacks, the attackers typically get the initial access in a more exposed machine, and need to move laterally to other machines holding critical data of the target organization. After the initial access, attackers increasingly target Microsoft Active Directory for obtaining critical domain credentials, and hence gaining the ability to move laterally in the victim

network. Leveraging stolen domain credentials is the most popular way for attackers to move across a target network, accounting for 80% of modern cyberattacks [37]–[41]. Half of organizations worldwide have experienced such an attack in recent years, while 40% of those attacks were successful [40], [42]. These two major limitations are further described in Chapter 4 in detail.

To address the first limitation of existing PIDS, we create Cyber Persistence Detector (Cpd) [18], the first dedicated persistence detection system, enabling multi-phase APT detection and investigation. Cpd is founded on the insight that persistent operations typically manifest in two phases: the "persistence setup" and the subsequent "persistence execution". By causally relating these phases, we enhance our ability to detect persistent threats. First, Cpd discerns setups signaling an impending persistent threat and then traces processes linked to remote connections to identify persistence execution activities. A key feature of our system is the introduction of *pseudo-dependency edges* (pseudo-edges), which effectively connect these disjoint phases using data provenance analysis, and *expert-guided edges*, which enable faster tracing and reduced log size. These edges empower us to detect persistence threats accurately and efficiently. Moreover, we propose a novel alert triage algorithm that further reduces false positives associated with persistence threats. Besides, we find that state-of-the-art deep learning-based systems [35], [36] fail to learn persistence attacks' semantics, not to mention symbolic machine learning approaches [43], [44], as they are restricted to learn local context. Evaluations conducted on well-known datasets demonstrate that our system reduces the average false positive rate by 93% compared to state-of-the-art methods.

To address the second limitation of existing PIDS, we introduce Hades [19], the first PIDS capable of performing accurate causality-based cross-machine tracing by leveraging a novel concept called *logon session-based execution partitioning and tracing* to overcome several challenges in cross-machine tracing. Hades's cross-machine tracing relies on the presence of an identity and access management (IAM) system, like Microsoft Active Directory, in an enterprise network. IAM systems are ubiquitous in modern enterprises' IT infrastructures, with the most popular one being Active Directory. 90% of Fortune 1000 companies rely on Active Directory, making it a top target of APT actors [40], [45]. We design Hades as an efficient on-demand tracing system, which performs whole network tracing only when it first identifies an authentication anomaly signifying an ongoing Active Directory attack, for

which we introduce a novel lightweight authentication anomaly detection model rooted in our extensive analysis of Active Directory attacks conducted by APT actors. To triage Active Directory attack alerts, we present a new algorithm integrating two key insights we identified in Active Directory attacks. Our evaluations on well-recognized MITRE's emulation plans [46] show that HADES outperforms not only popular open-source detection systems but also a prominent commercial Active Directory attack detector.

Finally, we present our third system COMMANDER [47], the first system capable of performing cross-machine multi-phase APT detection in industrial-sector organizations. Our analysis of APT threat reports linked in the MITRE ATT&CK knowledge base [48] yields the finding that HADES is susceptible to several evasive attack tactics/techniques routinely employed by APT actors: persistence, session hijacking, and port forwarding. Whereas port forwarding attacks can directly interrupt HADES's cross-machine tracing, both persistence and session hijacking attacks hinder accurate intra-machine tracing, on which cross-machine tracing hinges.

Recognizing the weaknesses of HADES, COMMANDER integrates CPD with HADES, and incorporates another two specialized detectors for session hijacking and port forwarding, respectively. These specialized detectors make complementary contributions to ensuring robust and correct whole network tracing, by delivering critical information to guide and adjust the tracing process. Further, COMMANDER is designed as a PIDS for detecting cross-machine attacks against industrial-sector organizations consisting of an enterprise network and an industrial control system (ICS), which are typically in two separate domains. That is, COMMANDER is able to trace across domains, given that the enterprise network and the ICS network are often connected via a gateway server. Besides, we developed parsers for logs from two popular industrial controllers in energy systems, i.e., Siemens SIPROTEC [49] and Hitachi Energy RTU500 [50], and detection rules with a reference to the MITRE ATT&CK Matrix for ICS [51] for these controllers. COMMANDER is capable of accurately attributing the activities on these controllers to the true identity behind those actions, even if the access originates from the enterprise networks. Our evaluations on MITRE's emulation plans extended by this work show that COMMANDER accurately detects attacks, outperforms open-source detection rules and a commercial detection system, and produces succinct and informative attack graphs.

## 1.1 Methodology

In this thesis, as the first step, we survey existing detection and investigation systems for APT attacks, particularly PIDS, in order to gain an overview of state-of-the-art systems. Meanwhile, we extensively analyze threat reports linked in the MITRE ATT&CK knowledge base [48] and posted in leading security vendors' websites, like CrowdStrike [37], Mandiant [52], Microsoft [53], Dragos [54], Trellix [55]. Our investigation of both existing security systems and real-world APT attacks yields the finding that there is a notable discrepancy between attack scenarios addressed in existing security systems and real-world APT attacks. That is, existing security systems focus on detecting and investigating simplified APT attacks, i.e., single-machine Advanced Non-Persistent Threat attacks, as opposed to real-world APT attacks, in which both cross-machine and persistence attack stages are very common. Based on our observation, we further review all existing datasets for research on APT detection and investigation that we can find. Our conclusion that existing security systems target only simplified APT attacks is consistent with the lack of attack authenticity in existing datasets' attack scenarios.

Motivated by these findings, we aim to design novel systems for detecting and investigating real-world APT attacks, i.e., cross-machine multi-phase APT attacks. As these sophisticated APT attacks pose several big, yet distinct challenges for detection and investigation, we first divide the task into two subtasks: multiple-phase APT detection and investigation, cross-machine APT detection and investigation. Then we explore methods for robust cross-machine multiple-phase APT detection and investigation. At the same time, we search for alternatives for evaluating our systems, leading to the discovery of MITRE emulation plans [46]. These emulation plans closely reproduce APT groups' real-world attack campaigns observed in the past. As MITRE has not published any datasets, we resort to stringently implementing these emulation plans.

## 1.2 Research Questions

The central research questions that this thesis is rooted at are: *How well can existing systems perform in detecting and investigating cross-machine multi-phase APT attacks? What can be improved, and how?* In the pursuit

of improvements over existing systems, a series of main research questions have emerged and are addressed in this thesis. These research questions are as follows:

***RQ1*** Are the existing datasets that are frequently used for evaluating state-of-the-art APT detection and investigation systems really suitable for the evaluation?

- ***RQ1.1*** How sound are the existing datasets frequently used for evaluating state-of-the-art APT detection and investigation systems? (Chapter 4)
- ***RQ1.2*** Are the attack scenarios in the existing datasets authentic enough comparing to real-world APT attacks? If not, what are the alternatives? (Chapter 4 and 6)
- ***RQ1.3*** Are there any criteria for assessing the quality of a dataset for research on APT detection and investigation defined? If not, what should the most important criteria be? (Chapter 6)

***RQ2*** What makes a sound system for detecting and investigating Multi-Phase APT attacks?

- ***RQ2.1*** How widespread is APT actors' use of persistence techniques in real world, and what are the consequences? (Chapter 4)
- ***RQ2.2*** How well can existing systems including SIEM detection rules, prior PIDS, and commercial detection systems perform in detecting persistence techniques and hence multi-phase APT attacks? (Chapter 7)
- ***RQ2.3*** What are the critical steps and insights in accurately and efficiently detecting persistence techniques and multi-phase APT attacks? (Chapter 5 and 7)

***RQ3*** What makes a sound system for detecting and investigating Cross-Machine APT attacks?

- ***RQ3.1*** How prevalent are cross-machine APT attacks, and what types of cross-machine APT attacks are there? (Chapter 4 and 8)
- ***RQ3.2*** How do APT actors conduct cross-machine attacks, and do they rely on misusing system functionalities/infrastructures? (Chapter 4 and 8)

***RQ3.3*** How well do existing systems perform in detecting cross-machine APT attacks and assisting the investigation? (Chapter 8)

***RQ3.4*** What are the critical steps and insights in accurately and efficiently detecting cross-machine APT attacks? (Chapter 5 and 8)

***RQ4*** What makes a sound system for detecting and investigating Cross-Machine Multi-Phase APT attacks?

***RQ4.1*** Can the detection and investigation system for cross-machine APT attacks introduced in Chapter 8 be evaded by persistence techniques, or any other attack techniques frequently employed by APT actors? (Chapter 4)

***RQ4.2*** What are the critical steps and insights in accurately and efficiently detecting cross-machine multi-phase APT attacks? (Chapter 5 and 9)

***RQ4.3*** Are provenance-based detection and investigation systems suitable for APT attacks against industrial-sector organizations, and do they provide value in attack recovery and remediation? (Chapter 9)

***RQ4.4*** Can an APT actor's traversal inside an organization be accurately assessed via provenance analytics, or can the attack activities on a field device in the ICS network be correctly attributed to the initial access in the enterprise network? (Chapter 9)

## 1.3 Contributions of the Present Work

In the present work, we created a sound, authentic dataset for research on APT detection and investigation, and developed three APT detection and investigation systems, enabling multi-phase APT detection and investigation, cross-machine APT detection and investigation in enterprise networks, and cross-machine multi-phase APT detection and investigation in industrial-sector organizations, respectively.

In addressing ***RQ1.1*** - ***RQ1.3***, we first identify several flaws of existing datasets, in particular a significant gap between existing datasets' attack scenarios and real-world APT attacks. To support our claim, we provide a

detailed comparison of prior datasets for research on APT detection and investigation in Chapter 6. It is worth noting that, although this comparison does not include unpublished datasets used to evaluate prior systems like [23], [24], [26], our investigation of the attack scenarios discussed in the corresponding articles' evaluation sections confirms that these attack scenarios have the same limitations. Due to the absence of criteria for assessing dataset quality, we propose an initial set of quality criteria. In addition, we present a novel sound dataset called AVIATOR, backed by MITRE emulation plans. Unlike most existing datasets, we provide log shipping tools and log parsing tools along with our dataset for increased dataset operability & usability. We also provide the logging configuration files for enhanced dataset reproducibility & extensibility.

In order to answer ***RQ2.1***, we conduct in Chapter 4 a thorough analysis of persistence attack techniques employed in real world, and shed light on the popularity of persistence techniques among APT actors. In addressing ***RQ2.2***, both our theoretical analysis and empirical evaluation show that existing systems are ill-suited for persistence detection, and unable to detect multi-phase APT attacks. Hence, we propose CPD in Chapter 7, which is, to the best of our knowledge, the first detection system specifically targeting persistence threats. In CPD, we introduce pseudo-dependency edges to causally relate disjoint persistence phases and improve the detection of these threats. Further, we propose the novel concept of expert-guided edges to enable efficient provenance tracing of persistence threats. On top of that, we identify critical insights on determining malicious persistence behaviors and propose an alert triage algorithm incorporating these insights to reduce false positives. The design of CPD provides an answer to ***RQ2.3***. We implement and evaluate CPD on diverse datasets, demonstrating better attack detection rates and provenance graph completeness versus state-of-the-art methods.

In Chapter 4 and 8, we survey cross-machine APT attack types and their popularity, and discover that most cross-machine APT attacks are based on exploiting specific Active Directory functionality, answering ***RQ3.1*** and ***RQ3.2***. In addition, we present a succinct and contextualized Active Directory attack overview, critical for understanding and detecting Active Directory-based cross-machine attacks. To deliver an answer for ***RQ3.3***, we perform both theoretic analysis and empirical evaluation on prior detection and investigation systems in Chapter 8. Motivated by the poor performance of prior systems, we propose HADES, whose design provides an answer for ***RQ3.4***. In HADES, we first introduce a lightweight authentication anomaly detection model for

Active Directory-based cross-machine attacks. Further, we propose a novel concept called logon session-based execution partitioning and tracing, and demonstrate the first accurate and efficient causality-based cross-machine provenance tracing system. On top of that, we present a new alert triage algorithm for accurate Active Directory-based cross-machine APT detection and investigation.

Subsequently, to investigate HADES's robustness and hence answer ***RQ4.1***, we introduce several attack techniques for evading HADES's cross-machine tracing, namely persistence, session hijacking, and port forwarding, in Chapter 4. In addressing ***RQ4.2***, we present two specialized detectors for session hijacking and port forwarding attacks, respectively, as a countermeasure in Chapter 9. In addition, we propose COMMANDER, a modularized provenance-based detection and investigation system that integrates CPD and the two specialized detectors with HADES to counter evasion attacks. To answer ***RQ4.3***, we evaluate COMMANDER on two extended MITRE emulation plans with attack steps on ICS. We develop parsers and TTP (Tactics, Techniques, Procedures) detection rules for two popular industrial controllers. Finally, we demonstrate COMMANDER's ability to perform robust whole network tracing, including attribution of malicious activities conducted on industrial controllers, which delivers an answer for ***RQ4.4***.

## 1.4 Thesis Outline

This thesis is structured as follows:

- Chapter 2 briefly presents topics closely related to this thesis, including MITRE ATT&CK, auditing & logging, TTP-based hunting.
- Chapter 3 discusses related research studies, and introduces a taxonomy of APT detection and investigation systems. It also describes the threat model of our systems.
- Chapter 4 demonstrates the major problems addressed in this thesis. These problems are identified during our examination of existing detection and investigation systems for APT attacks, and extensive analysis of threat reports linked in the MITRE ATT&CK knowledge base and posted in leading security vendors' websites.

- Chapter 5 provides an overview of our approach, in which we briefly describe how we successively and progressively solve the problems introduced in the last Chapter.
- Chapter 6 further highlights the flaws in existing datasets for research on APT detection and investigation, and the merits of MITRE emulation plans. Our dataset AVIATOR, which is derived from the original and extended MITRE emulation plans, is compared with prior datasets in this chapter.
- Chapter 7 describes concrete persistence attack techniques, and resulted multiple-phase APT attacks. In this chapter, we introduce CPD and demonstrate its superior performance in persistence detection to existing detection systems.
- Chapter 8 further analyzes Active Directory-based cross-machine APT attacks. We present the design and evaluation of our second APT detection and investigation system HADES in this chapter.
- Chapter 9 addresses cross-machine multiple-phase APT detection and investigation in industrial-sector organizations with our third system COMMANDER.
- Chapter 10 concludes this thesis and points out future research directions.

## 1.5 Publications

> **Note:** In the following, to differentiate our own works from others, we cite our own works with a leading letter "P".

In [P1] and [P2], we investigate rule learning-based approaches for intrusion detection. However, we find that these approaches are restricted to learning local context, and can hardly be extended to learn the global context required for differentiating between malicious and benign behaviors. This limitation is, however, not restricted to symbolic machine learning approaches. It applies also to subsymbolic machine learning approaches, as we show in Chapter 7 that Graph Neural Networks (GNN)-based approaches fail to learn the global context necessary for detecting APT attack stages like persistence.

In [P3] we conduct research on binary exploitation detection and prevention, which is often employed for initial access, the first step in a Cyber Kill Chain. However, we find that binary exploitation detection requires logs of execution of basic blocks or even machine instructions, and long-term logging at this level produces massive data volumes impossible to store in practice. Even then, reliable binary exploitation detection remains a hard problem. Rather, preventive defense techniques like data execution prevention [56], control-flow integrity [57], and address space layout randomization [58] are employed to counter binary exploitation in reality.

That is, from the detection and investigation point of view, APT attacks can be more reliably exposed at the later attack stages, instead of the initial access attack stage. To detect the later attack stages, system logs collected at the system call level are often employed. Logging at the system call level strikes a balance between the semantic level of logs and the cumulative log size, as further discussed in Section 2.5. Therefore, by leveraging system logs, we design three APT detection and investigation systems for post-exploitation Cyber Kill Chain conducted by APT actors. Our first system [P4] addresses persistence techniques and hence multi-phase APT attacks, whereas our second system [P5] targets Active Directory-based cross-machine APT attacks. Our last system [P6] further strengthens the defense against cross-machine multi-phase APT attacks. Through the development of our APT detection and investigation systems, we also created a sound and authentic dataset in [P7] based on MITRE emulation plans.

[P1] Q. Liu, V. Hagenmeyer, and H. B. Keller, "A review of rule learning-based intrusion detection systems and their prospects in smart grids", *IEEE Access*, vol. 9, pp. 57 542–57 564, 2021. DOI: 10.1109/ACCESS.2021.3071263 (equivalent to [43])

[P2] Q. Liu, H. B. Keller, and V. Hagenmeyer, "A bayesian rule learning based intrusion detection system for the MQTT communication protocol", in *Proceedings of the 16th International Conference on Availability, Reliability and Security*, 2021. DOI: 10.1145/3465481.3470046 (equivalent to [44])

[P3] Q. Liu, K. Bao, and V. Hagenmeyer, "Binary exploitation in industrial control systems: Past, present and future", *IEEE Access*, vol. 10, pp. 48 242–48 273, 2022. DOI: 10.1109/ACCESS.2022.3171922 (equivalent to [59])

[P4] Q. Liu, M. Shoaib, M. U. Rehman, K. Bao, V. Hagenmeyer, and W. U. Hassan, *Accurate and scalable detection and investigation of cyber persistence threats*, 2024. arXiv: 2407.18832 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2407.18832, under review (equivalent to [18])

[P5] Q. Liu, K. Bao, W. U. Hassan, and V. Hagenmeyer, *HADES: Detecting Active Directory attacks via whole network provenance analytics*, 2024. arXiv: 2407.18858 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2407.18858, under review (equivalent to [19])

[P6] Q. Liu, K. Bao, and V. Hagenmeyer, "COMMANDER: A robust cross-machine multi-phase Advanced Persistent Threat detector", under review (equivalent to [47])

[P7] Q. Liu, K. Bao, and V. Hagenmeyer, "AVIATOR: A MITRE emulation plan-derived living dataset for Advanced Persistent Threat detection and investigation", in *Proceedings of 2024 IEEE International Conference on Big Data*, 2024, in print (equivalent to [60])

# 2 Preliminaries

In this chapter, we introduce topics closely related to this thesis, including Advanced Persistent Threat, MITRE ATT&CK, auditing & logging, TTP-based hunting, provenance-based detection and investigation systems.

## 2.1 Advanced Persistent Threat & Cyber Kill Chain

Advanced Persistent Threat (APT) actors are well-resourced and skilled attackers who specialize in launching covert cyberattacks on computer systems to which they maintain unauthorized access for a prolonged period of time [1], [2]. These attackers typically employ social engineering or binary exploitation [59], [61] to gain the initial access. This first-time access may allow for continued access, or be of limited use due to, e.g., credential changes or software updates [62].

As the main objective of APT actors is cyber espionage [4]–[8], or impairing physical processes at a *strategic* point when targeting critical infrastructures [9]–[13], they mostly do not cause immediate destructive damages to the targeted organizations after the initial access, like network outage caused by spreading computer viruses. Instead, they aim to monitor the victim organizations to find, intercept, and exfiltrate sensitive data. That is, the hallmark of APT actors is not opportunistic hit-and-run tactics, but rather maintaining a low-and-slow strategy, i.e., being stealthy and remaining undetected for as long as possible after the initial foothold in the victim networks.

To model the attack behaviors of APT actors, Lockheed Martin coined the term Cyber Kill Chain (CKC) by adapting the military phrase Kill Chain [3]. As per [3], a CKC is a systematic attack process including several attack stages: Reconnaissance, Weaponization, Delivery, Exploitation, Installation, Command and Control (C2), and Actions on Objectives. Put in a concrete example, an attacker first conducts some *reconnaissance* by crawling the

Internet to find any useful information about the target organization, e.g., email addresses. Then the attacker *weaponizes* a Microsoft Office document by inserting some malicious macro code, before *delivering* the document to employees of the target organization via email. The malicious macro code *exploits* some program or configuration vulnerability to *install* a *command and control* agent program. Finally, the attacker is able to *act* directly on the victim system via the C2 agent program.

## 2.2 Living Off the Land Binaries

One of the most important strategies applied by APT actors to remain undetected is routinely employing Living-Off-the-Land Binaries (LOLBins) [63], [64]. LOLBins [15]–[17] include system built-in programs and user-installed programs, which are often used by system administrators and normal users themselves. Intuitively, leveraging these programs helps to evade detection due to their legitimate nature. That is, the execution of LOLBins and the associated system activities are often considered as benign by detection systems without further scrutiny due to the sheer amount of system activities caused by them during normal operation.

A concrete example is `powershell.exe`, which is a pre-installed program deeply integrated into the Windows OS. Through `powershell.exe`, system administrators can automate nearly any administrative tasks on systems managed by them. Hence, APT actors commonly misuse `powershell.exe` for malicious purposes and easily "hide in plain sight". In the aforementioned CKC example, the attacker could write some Powershell code acting as the C2 agent instead of actually dropping a program file and installing it. By doing so, the attacker evades antivirus programs that scan for creation of malicious files and installation of malicious programs.

Another example is `setspn.exe`, which is a built-in system program on Windows for performing Active Directory discovery. In comparison to network scanning using third-party tools like nmap [65], SPN (Service Principal Name) scanning via `setspn.exe` is much less noisy while achieving the same goal, i.e., identifying potentially vulnerable servers in the network. Whereas only some LOLBins like `powershell.exe` and `wmic.exe` can be used for various attack stages, others are more or less restricted to a single attack stage/purpose. For instance, programs like `systeminfo.exe` and `whoami.exe` can only be used for

local discovery purpose, and the program `bitsadmin.exe` is typically used by attackers to covertly exfiltrate data out of victim systems.

## 2.3 MITRE ATT&CK

The MITRE ATT&CK Matrix [20] is a knowledge base of attack tactics and techniques based on real-world observations. This Matrix is curated by the MITRE Corporation, which is one of the most-known organizations in the security industry. On the one hand, the Matrix is maintained with contributions from industrial security vendors. On the other hand, security vendors use this Matrix as a reference for further creating detection rules. There are in fact three MITRE ATT&CK matrices, applied to Enterprise [20], Mobile [66], ICS [51], respectively. However, the MITRE ATT&CK Matrix is almost always referred to the matrix for Enterprise [20].

These matrices present the so-called TTP, which stands for Tactics, Techniques, Procedures. Whereas tactics present very high-level summarization, techniques are concrete attack steps, and procedures are a specific implementation of an attack technique. The attack stages introduced in the original CKC [3] can be loosely mapped to the attack tactics in the ATT&CK Matrix. As the MITRE ATT&CK Matrix includes more attack stages and is being actively maintained, and hence more complete and up-to-date, we use phrases from the MITRE ATT&CK Matrix instead of the ones introduced by Lockheed Martin [3] to label attack stages in a CKC in the present work.

## 2.4 Cyber Kill Chain in ICS

Based on our lab infrastructure, we illustrate a Cyber Kill Chain on an industrial control system (ICS) in Figure 2.1. In a typical APT attack campaign, attackers gain *initial access* to a victim organization mainly through program exploits or stolen credentials [62]. Program vulnerabilities are often patched, and users may change passwords, making these methods unreliable for long-term operations. Thus, attackers *establish persistence* using various methods for prolonged, reliable access. Post-persistence, they perform *local discovery* to understand current systems, and conduct *network discovery* to find and *move laterally* to other vulnerable machines. To *evade detection*,

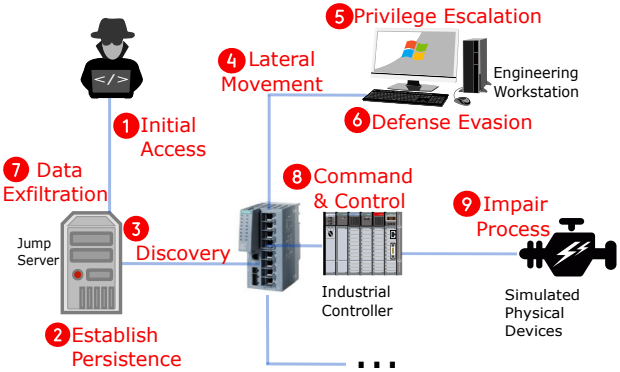



**Figure 2.1:** Cyber Kill Chain on an industrial control system.

they select tools to bypass the security programs or deactivate them after *privilege escalation*. The hallmark of APT campaigns is not immediate impact but remaining undetected for a long period, aiming to *collect* and *exfiltrate data*, and/or *command & control* the industrial controllers and *impair physical processes* at a strategic point. The ability to achieve persistence is critical for attackers' success.

For the sake of simplicity, in Figure 2.1, we describe the Cyber Kill Chain only on the ICS network part. In practice, a Cyber Kill Chain conducted by APT attackers against an industrial-sector organization typically starts from the enterprise network, passes through the operational technology (OT) network and ends at the field device network, as shown in Figure 2.2. The pivotal access on a gateway server enables attackers to reach machines in the OT network. Due to network segmentation, cross-machine activities / lateral movements are necessary for attackers to reach Engineering Workstations, through which field controllers are programmed and configured, and field controllers, like PLC (Programmable Logic Controller), RTU (Remote Terminal Unit) and IED (Intelligent Electronic Device), through which physical processes are controlled.

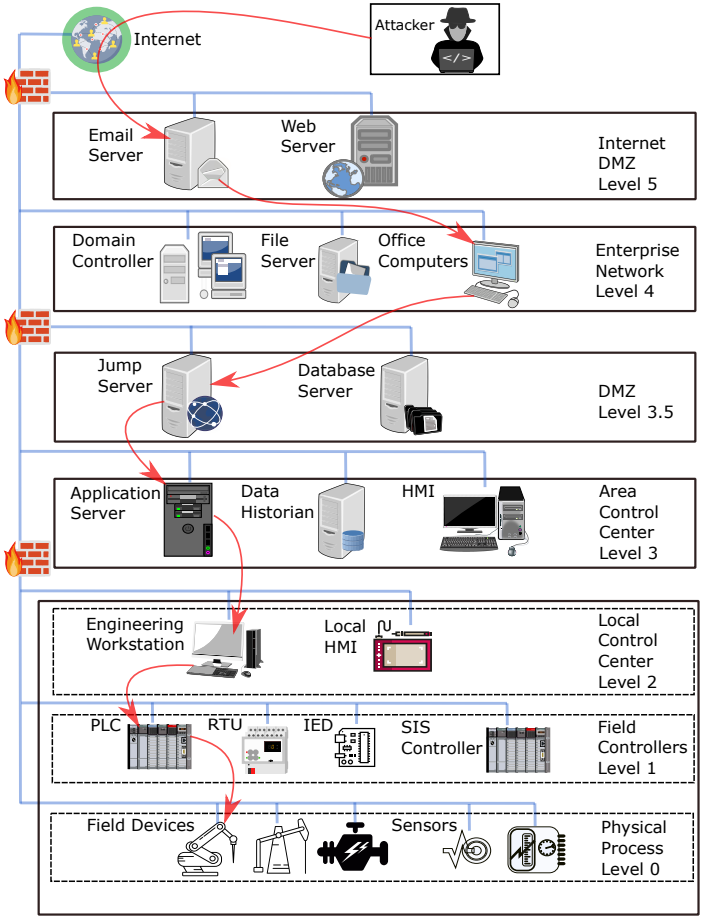



**Figure 2.2:** An example of a complete attack path on an industrial-sector organization [P3].

## 2.5 Auditing & Logging

One of the biggest challenges in APT detection is the difficulty of knowing in advance what data from which abstraction level are required. Simply collecting extensive data is inefficient and impractical, or even counterproductive. Automatically accurately differentiating an attack from benign behaviors

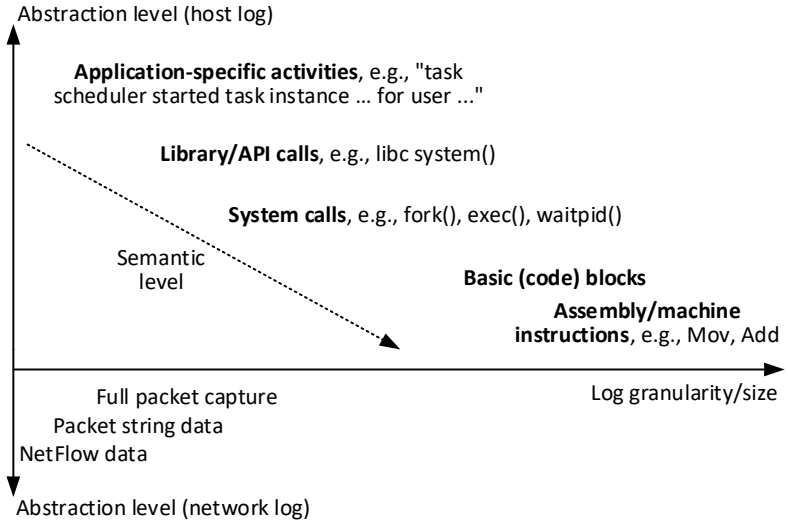



**Figure 2.3:** Logging at different abstraction levels [P7].

is non-trivial, as attack-related behaviors and benign behaviors could often appear identical based on the chosen data. To discern the attack, additional context may be needed from data not chosen previously or, even worse, not collected by defenders.

Both network logs and host logs can be collected at different abstraction levels, as depicted in Figure 2.3. Among different types of network logs, full packet capture data, aka PCAP, contain the most detailed data, whereas network flow data only include metadata of the PCAP. Network logs are typically ingested in network-based intrusion detection systems (NIDS). Comparing to network logs, host logs provide a more complete and true picture of what has happened in the systems. Logging in hosts is, however, more complicated, and requires more consideration and better understanding of attacks being investigated. The lower abstraction level the logs are collected at, the more fine granular the logs are, better reflecting what has really happened and assisting attack detection and investigation. However, the more fine granular the logs are, the bigger the cumulative log size will be, and thus the sooner the logs have to be deleted due to a limit of log storage capacity in practice.

Logs of execution of basic blocks or machine instructions can be used for detection of binary exploitation. However, long-term logging at this level

causes intolerable amount of data, and yet has unreliable detection results [59]. Rather, preventive defense techniques like code-pointer integrity [67], data execution prevention [56], control-flow integrity [57], and address space layout randomization [58] are typically employed to counter binary exploitation. Logging at the system call level strikes a balance between the semantic level of logs and the cumulative log size. In fact, the DARPA Transparent Computing program [68] determines that logs from the system call level provide the details necessary for APT detection and investigation. This leads to three most popular DARPA datasets [69]–[71] for research on APT detection and investigation, which consist of mainly system logs. Still, logs at higher abstraction levels can make complementary contribution to accurate and efficient APT detection and investigation, as prior studies [19], [27] show.

## 2.6 TTP-based Hunting

Conventional intrusion detection systems (IDS) often focus on isolated system events, and excel at identifying malware causing a burst of malicious behaviors. Facing APT actors frequently employing LOLBins and the so-called low-and-slow strategy, these IDS tend to miss those attacks and suffer from a high false negative rate. To overcome this, security vendors resort to the MITRE ATT&CK Matrix [20] as a reference for further creating detection rules.

**Listing 2.1:** Exemplary MITRE TTP detection rule for Discovery

```
IF EXISTS event WHERE event.action = 'read' AND event.object =
    '/etc/issue' THEN ALERT ('Discovery, T1082')
```

**Listing 2.2:** Exemplary MITRE TTP detection rule for Command & Control

```
IF EXISTS event WHERE event.action = 'network connection' AND
    event.subject = 'svchost.exe' AND event.sourcePort = '3389'
     AND event.destinationPort : ('80', '443') THEN ALERT ('
    Command & Control, T1572')
```

Recent commercial detection systems like Symantec [72] and McAfee [73] produce security alerts with a TTP (Tactics, Techniques, Procedures) label, if monitored system activities can be mapped to a TTP in the MITRE ATT&CK

Matrix [20]. So do detection rules of open-source detection rule repositories like Sigma [74] and Google Chronicle [75]. Examples of detection rules for "Discovery" attack stage and "Command & Control" attack stage are given in Listing 2.1 and Listing 2.2, respectively.

However, there is a semantic gap between TTP alerts and malicious activities. Because not only malicious activities but also benign system activities can trigger a TTP alert. That is, if some system activity can be mapped to a TTP, it does not mean that it is certainly malicious. TTP-based detection/hunting alone inevitably exacerbates the alert fatigue problem, in which security programs create many more alerts than security analysts can ever investigate in practice. This is due to the fact that APT actors frequently make use of LOLBins to blend their malicious system activities into benign system activities, and it is often not possible to differentiate between them by looking at these system activities in isolation.

## 2.7 Tyranny of security alerts

On average, a security analyst spends 30 minutes investigating a single alert [76], [77]. Up to 99% of these alerts are false positives [21], leading to *alert fatigue*. A common response to the overwhelming number of unmanageable alerts is either setting thresholds or disabling high-volume alert rules. As a result, over two-thirds of alerts are never addressed [78], [79]. Effective and automatic reduction of alert volume is crucial for the functionality of a security operation center (SOC).

## 2.8 Provenance-based APT Detection and Investigation

System activities monitored for APT detection and investigation include every process creation, file/Registry access and network socket access, which can be put in the form of a graph, in which vertices represent system entities and edges represent causal dependencies. System entities include processes, files / Registry keys and network sockets, shown as rectangles, ovals, and diamonds in Figure 2.4, respectively. To really make use of the TTP in the MITRE

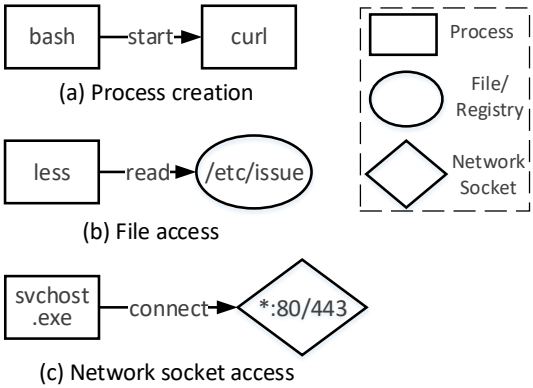



**Figure 2.4:** System activities in the form of a graph.

ATT&CK Matrix [20], more context is required to differentiate and prioritize TTP alerts for attack detection and investigation. Provenance-based intrusion detection systems (PIDS) emerged as a promising approach to realize the contextualization, differentiation and prioritization of TTP alerts, as they identify TTP alerts in a provenance graph, apply backward and forward tracing to connect related TTP alerts, resulting in reconstruction of an entire Cyber Kill Chain. Existing PIDS [22]–[26], [29], [33]–[36], [80]–[82] have already proven to be very effective in reducing false alarms, and detecting true attacks. The underlying assumption is that, when several TTP alerts can be causally linked into an attack graph reassembling a Cyber Kill Chain, as in Figure 2.5, they are much more likely caused by attackers than those alerts unrelated to each other.

Provenance-based security systems leverage logs collected at the system call level by popular auditing frameworks like Windows ETW (Event Tracing for Windows) [83], System Monitor [84], Linux Auditd [85], to detect and investigate APT attacks. Provenance graphs created with system logs present historical context of a system event, and demonstrate causal relations between system entities. Given an indicative event as a starting point, a backward tracing and forward tracing in the provenance graph can disclose more related malicious system activities, i.e., the root cause and attack ramifications, respectively. The objective of provenance graphs is to link an attacker's system activities that appear to be benign individually, but expose a malicious purpose collectively. The ability of PIDS to causally connect individual,

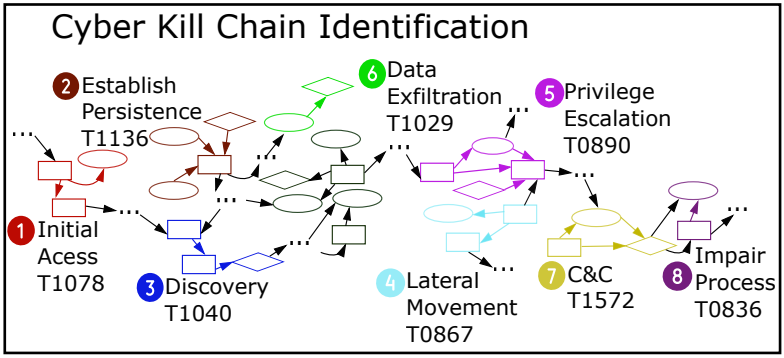



**Figure 2.5:** Identification of APT attack stages in a provenance graph.

seemingly benign system activities of attackers makes them significantly outperform other APT detection and investigation systems. The resulting context-rich attack graphs are often self-explanatory for security analysts, if pruned properly.

## 2.9 Evaluation Datasets

During our research on provenance-based APT detection and investigation systems, we experienced the difficulty of finding appropriate public datasets to evaluate our systems. We consider a dataset as appropriate for a detection system, if the dataset 1) is derived from attack scenarios in which the attack stages that the detection system is designed to detect are actually conducted, and 2) contains the logs required to detect the attack stages. In order to evaluate the performance in persistence detection of our first APT detection and investigation system CPD [18], an evaluation dataset must contain at least a persistence attack stage. However, to our surprise, the majority of public datasets for APT research do not include persistence attack stages, partly due to a common misunderstanding of persistence, as discussed in Chapter 4 in detail. Worse yet, Active Directory-based cross-machine attacks, which our second APT detection and investigation system HADES [19] aims to detect, are present in only one public dataset we could find, but this dataset does not include logs deemed necessary for accurate detection and investigation of cross-machine attacks.

In contrast, the MITRE emulation plans [46] include the most complete APT attack chains, featuring persistent, Active Directory-based cross-machine attacks. These emulation plans are created to evaluate the detection performance of commercial security solutions for attacks conducted in the style of specific APT groups, allowing more realistic assessment. Hence we evaluate each our system introduced with this thesis on at least two MITRE emulation plans. For instance, we evaluate CPD [18] on APT29 [86] and Sandworm [87] emulation plans, as only these two APT groups out of all APT groups in the MITRE emulation plans belong to the top 10 "persistent" APT groups (Table 4.2). We evaluate HADES [19] on APT29, WizardSpider [88] and Oilrig [89] emulation plans, as these emulation plans feature the most intensive cross-machine attacks. We evaluate COMMANDER [47] on our extended Oilrig and Sandworm emulation plans, as these two APT groups are most known for their attacks on industrial control systems. In Chapter 6, we introduce the dataset AVIATOR we created along with our three APT detection and investigation systems CPD [18], HADES [19], and COMMANDER [47]. In its current version, our dataset AVIATOR is most suitable for research on provenance graph-based APT detection and investigation.

# 3 Related Work & Threat Model

This chapter presents related research studies, in which we first introduce and categorize relevant existing systems, and then discuss closely related systems in detail. Additionally, we describe the threat model of our systems at the end.

## 3.1 Related Work

In Figure 3.1, we present a taxonomy of APT detection and investigation systems, which we categorize into main systems and auxiliary systems. Intuitively, systems designed directly for APT detection and investigation are considered as main systems, while systems that serve to assist the detection or investigation process, but do not perform detection themselves by producing attack alerts and/or attack graphs, are considered as auxiliary systems. We further categorize main systems into network log-based and host log-based systems, depending on the data source. In addition, we classify a detection system as single attack stage detector, if it is designed for detecting and investigating a specific attack stage, e.g., data exfiltration. In contrast, kill chain detectors have a broader detection scope: a Cyber Kill Chain consisting of several attack stages.

This chapter is based on a previous paper of ours. Note that, to differentiate our own works from others, we cite our own works with a leading letter "P". Inside the chapter, these kind of citations are used in particular for tables and figures taken or adopted from our previous paper:

[P6] Q. Liu, K. Bao, and V. Hagenmeyer, "COMMANDER: A robust cross-machine multi-phase Advanced Persistent Threat detector", under review

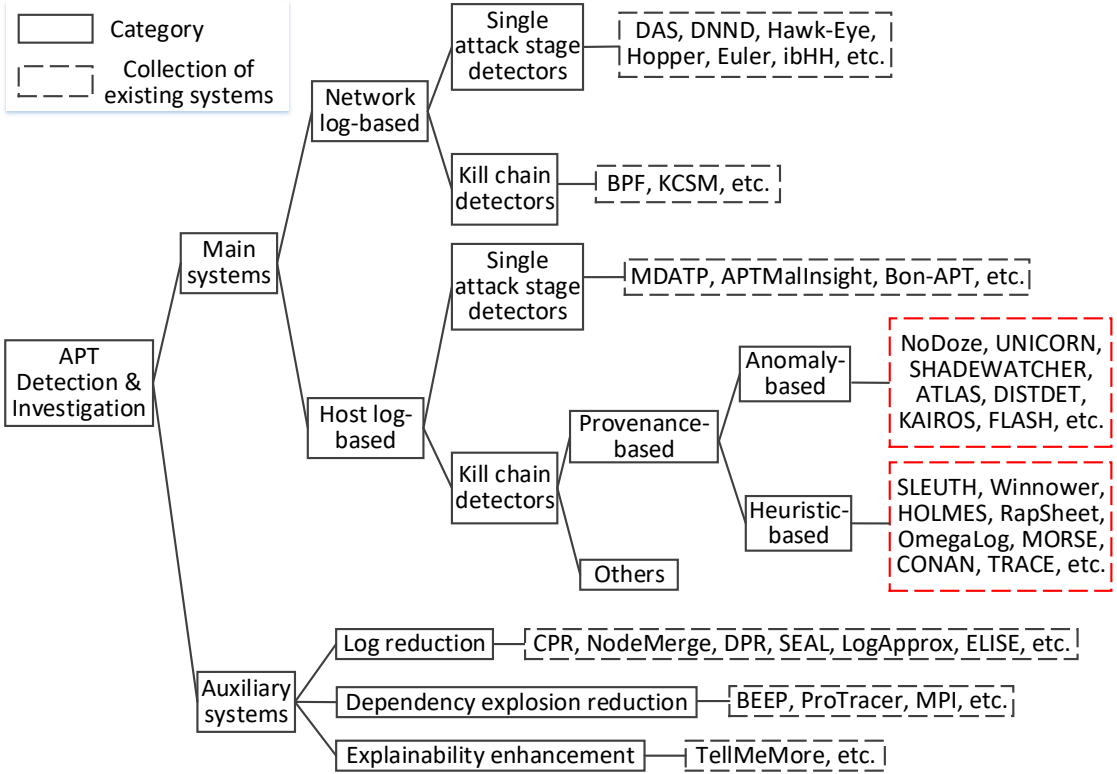



**Figure 3.1:** Taxonomy of APT detection and investigation systems [P6].

Our analysis of existing systems yields the finding that network log-based APT detection and investigation systems largely fall into the category of single attack stage detector, whereas host log-based APT detection and investigation systems mostly aim to detect an entire Cyber Kill Chain. For instance, Ho et al. introduced Hopper [90], and King et al. developed Euler [91], both targeting lateral movement detection using network logs. There are also network log-based single attack stage detectors for detecting phishing emails / initial access, e.g., DAS [92], command and control (C2) activities, e.g., DNND [93], Hawk-Eye [94], and data exfiltration, e.g., ibHH [95].

There are only a few network log-based kill chain detectors: BPF [96], KCSM [97]. By construction, network log-based kill chain detectors can only detect a partial Cyber Kill Chain, as some attack stages, e.g., persistence, credential access, and privilege escalation, can be detected only with host logs. BPF [96] proposes a framework based on belief propagation inspired from graph theory. It leverages DNS (Domain Name System) logs and web proxy logs to detect partial APT attack chains, and only considers malware delivery / initial foothold, and command and control attack stages. Besides, these attack stages are apparently detected separately, complicating the investigation process.

In contrast, another network log-based kill chain detector KCSM [97] can not only detect more attack stages, e.g., lateral movement and data exfiltration, but also produce attack graphs associating these attack stages, which provide security analysts helpful information for further investigating the attacks. KCSM operates by first processing Zeek [98] logs, and leveraging Graph-based Alert Correlation (GAC) [99] to generate high-level alerts, and finally contextualizing these alerts in attack scenario graphs. However, attack graphs output by KCSM [97] are very high-level attack graphs, in which nodes denote individual machines and edges denote suspected attack stages. To find the root cause for an attack or thoroughly assess the attack consequences, security analysts would still need to manually inspect host logs from each involved machine. That is, network log-based kill chain detectors like KCSM may be more efficient than host log-based ones in attack detection due to lower log requirements, but would fall short of fully assisting an attack investigation process.

We find that the majority of host log-based APT detection and investigation systems leverage data provenance analysis. Only a few detectors do not fall into the category of provenance-based intrusion detection systems

(PIDS), e.g., MDATP [100] for detection of credential access attack stage, and APTMalInsight [101] and Bon-APT [102] for detection of (APT Malware) execution attack stage. Comparing to network log-based kill chain detectors, host log-based kill chain detectors operate at a finer granularity. Attack graphs produced by PIDS expose malicious behaviors at the system activity level, e.g, malicious process creation or suspicious file access, rather than at machine level, e.g., suspicious network interaction.

A recent study [82] shows that PIDS are considered more effective than other detection systems, mostly due to better interpretability of detection results. Context-rich provenance graphs are leveraged in PIDS to accelerate attack detection & investigation process. PIDS are often categorized into anomaly-based systems and heuristic-based systems. Anomaly/learning-based PIDS model/learn benign behavior from provenance graphs, and detect deviations from it as anomalies. Heuristic-based PIDS embed expert knowledge and developed heuristics into detection algorithms. Heuristics are often derived from key insights identified during analysis of past real-world APT attack campaigns. We consider PIDS as the most related systems to our systems. These systems are marked in red, dashed bounding boxes in Figure 3.1, and further discussed in Section 3.1.1 in detail.

In order to enhance the practicality of provenance-based APT detection and investigation systems, a number of auxiliary systems have been proposed. For instance, various dedicated log reduction schemes are developed, e.g., CPR [103], NodeMerge [104], DPR [105], LogApprox [106], SEAL [107], ELISE [108]. They serve to reduce the volume of logs required for APT detection and investigation, and hence alleviate the long-term log storage problem. To realize log reduction, these systems leverage graph compression methods [107], [108], semantic pruning methods [104], information flow preservation methods [103], [105], causality approximation methods [106]. It is worth noting that, regardless of log reduction methods, there is a tradeoff between log reduction efficiency and forensic validity of reduced logs [109].

The large volume of host logs not only pose a log storage difficulty, but also deteriorate the dependency explosion problem. Dependency explosion happens for long-running processes, in which each input event is conservatively considered causally responsible for all subsequent output events, and vice versa, resulting in excessive amounts of false dependencies and formidably dense provenance graphs [110], [111]. Program execution partitioning techniques introduced in BEEP [110] and ProTracer [80] are the first proposed approaches

for combating dependency explosion. MPI [112] improves these techniques by introducing a semantic-aware program annotation and instrumentation technique.

With the increased popularity of anomaly/learning-based PIDS in recent years, Welter et al. realized the importance of enhancing the explainability of anomaly-based PIDS, and proposed an auxiliary system for explainability enhancement, i.e., TellMeMore [113]. It presents an input permutation-based approach, in which it first systematically modifies a whole provenance graph that is already classified as an anomaly by an anomaly-based PIDS, and then observes if the anomaly-based PIDS still classifies the modified whole provenance graph as an anomaly. Its objective is to identify a specific part of the graph, i.e., a set of nodes and edges, responsible for triggering an anomaly. This is helpful for attack investigation, as security analysts can better reason about an anomaly alert, and decide whether to further inspect it or dismiss it. However, TellMeMore [113] is only applicable to anomaly-based PIDS that label an anomaly to an entire provenance graph [28], [81], or a time window [35]. Other anomaly-based PIDS already can label an anomaly to a single edge [33], or node [36], [114], [115]. It is also worth noting that, despite recent improvements, anomaly-based PIDS still have lower interpretability than heuristic-based PIDS, as they may flag a large amount of nodes and/or edges as anomalous.

### 3.1.1 Prior PIDS

Table 3.1[1] provides a comparison of prior PIDS and our PIDS. SLEUTH [22] is a PIDS that introduces node tagging and tag propagation based on heuristics for attack graph reconstruction. It is designed for detecting and investigating simplified APT attacks, and focuses on attacks on individual hosts in isolation. We consider an attack scenario as simplified APT attack, if the entire scenario is restricted to a single host and does not contain an effective persistence technique, as apposed to real-world APT attacks. SLEUTH produces alerts as

[1] appli. = applications, APT = Advanced Persistent Threat, CM = Cross-Machine, Detect. = Detection, Distr. = Distributed, Enter. = Enterprise, Envir. = Environment, Eval. = Evaluation, Indiv. = Individual, Invest. = Investigation, IS = Industrial-sector, MP = Multi-Phase, ML = Machine Learning, N/A = not applicable/available, netw. = networks, optim. = optimization, organ. = organizations, Recomm. = Recommendation, Resp. = Response, scen. = scenario, SHADEW. = SHADEWATCHER, Simp. = Simplified, unpubli. = unpublished.

detection result, and succint attack graphs to support investigation by security analysts. Evaluation on a DARPA dataset [69] shows that SLEUTH has a short response time, i.e., within a few seconds or minutes. Winnower [23] is a heuristic-based PIDS that adapts graph grammar techniques to detect attacks on distributed applications, instead of whole systems. It is evaluated on a unpublished, self-developed dataset.

NoDoze [24] is an anomaly-based PIDS that leverages detection rules from commercial security products to generate initial alerts. Then it performs tracing on these alerts with system logs, and employs behavioral execution partitioning for enhanced tracing efficiency. With a path-based anomaly scoring algorithm, NoDoze is the first PIDS producing ranked alerts. Ranked alerts are more useful in practice, as security analysts can prioritize the highest ranked alerts for investigation due to limited time. It is worth noting that NoDoze's anomaly based approach assigns low score to common events caused by "benign" programs as it assumes that attackers mostly rely on malicious programs. This is, however, a weak assumption due to heavy use of LOLBins by APT attackers.

HOLMES [25], as a heuristic-based PIDS, proposes the notion high-level scenario graph (HSG), which is a summarized description of an attack history. It first produces initial alerts through self-developed detection rules, then performs tracing via correlation between suspicious information flows, and outputs HSG at the end. RapSheet [26] also combines detection rules with heuristics for APT detection and investigation. Instead of relying on self-developed detection rules, RapSheet employs a commercial EDR (Endpoint Detection and Response) product for the initial alerts. Unlike NoDoze, RapSheet adopts a graph-based scoring scheme for ranking final alerts. RapSheet also proposes techniques to perform graph reduction, which makes produced attack graphs more succinct. OmegaLog [27] presents the first approach realizing log unification, in which system logs and application logs are combined to enhance tracing efficiency, and to produce succinct attack graphs with enriched context.

**Table 3.1:** Comparison of PIDS [P6].

| Name | Methods | Rule sources | Innovative techniques | Tracing optim. | Attack scope | Target Envir. | Detect. result | Investigation assistance | Eval. Datasets | Resp. time |
|---|---|---|---|---|---|---|---|---|---|---|
| SLEUTH [22], 2017 | Heuristics | N/A | Node Tagging, Tag Propagation | ✓ | Simp. APT | Indiv. hosts | Alerts | Succinct attack graphs | DARPA | short |
| Winnower [23], 2018 | Heuristics | N/A | Graph Grammar Models | ✗ | Non-APT | Distr. appli. | Alerts | Succinct attack graphs | unpubli. | short |
| NoDoze [24], 2019 | Rules, Anomaly | commercial | Behavioural Execution Partitioning | ✓ | Simp. APT | Indiv. hosts | Ranked alerts | Succinct attack graphs | unpubli. | short |
| HOLMES [25], 2019 | Rules, Heuristics | self-devloped | Information Flow Correlation | ✗ | Simp. APT | Indiv. hosts | Ranked alerts | High-level scen. graphs | DARPA | short |
| RapSheet [26], 2020 | Rules, Heuristics | commercial | Graph-Based Scoring Scheme | ✓ | Simp. APT | Indiv. hosts | Ranked alerts | Succinct attack graphs | unpubli. | short |
| OmegaLog [27], 2020 | Heuristics | N/A | Log Unification, Peephole Concretization | ✗ | Non-APT | Indiv. hosts | Alerts | Succinct attack graphs | unpubli. | short |
| UNICORN [28], 2020 | ML, Anomaly | N/A | Sketch-based Provenance Encoding | ✗ | Simp. APT | Indiv. hosts | Alerts | None | DARPA, StreamSpot | long |
| MORSE [29], 2020 | Heuristics | N/A | Tag Attenuation, Tag Decay | ✓ | Simp. APT | Indiv. hosts | Alerts | Succinct attack graphs | DARPA | short |
| CONAN [30], 2020 | Heuristics | N/A | State-based Detection Framework | ✗ | Simp. APT | Indiv. hosts | Alerts | Succinct attack graphs | DARPA | short |
| TRACE [31], 2021 | Heuristics | N/A | Unit-based Selective Instrumentation | ✓ | Simp. APT | Enter. netw. | Alerts | Succinct attack graphs | DARPA | short |
| ATLAS [32], 2021 | ML, Anomaly | N/A | Sequence-based Model Learning | ✗ | Simp. APT | Small netw. | Alerts | Succinct attack graphs | ATLAS | long |
| SHADEWA. [33], 2022 | ML, Anomaly | N/A | Recomm.-guided Log Analysis | ✗ | Simp. APT | Indiv. hosts | Alerts | None | DARPA | N/A |
| DISTDET [34], 2023 | Anomaly | N/A | Alarm Deduplication & Semantic Aggregation | ✗ | Simp. APT | Small netw. | Ranked alerts | Succinct attack graphs | DARPA, unpubli. | N/A |
| KAIROS [35], 2024 | ML, Anomaly | N/A | Encoder-Decoder-based Anomaly Quantification | ✗ | Simp. APT | Indiv. hosts | Alerts | Verbose attack graphs | DARPA, StreamSpot | long |
| FLASH [36], 2024 | ML, Anomaly | N/A | Embedding Recycling Database | ✗ | Simp. APT | Indiv. hosts | Ranked alerts | Verbose attack graphs | DARPA, StreamSpot | long |
| CPD, 2024 | Rules, Heuristics | open-source, self-devloped | Pseudo Edges, Expert-guided Edges | ✓ | MP APT | Indiv. hosts | Ranked alerts | Succinct attack graphs | DARPA, AVIATOR | short |
| HADES, 2024 | Rules, Heuristics | open-source | Logon Session-based Execution Partitioning | ✓ | CM APT | Enter. netw. | Ranked alerts | Succinct attack graphs | AVIATOR | short |
| COMMANDER, 2024 | Rules, Heuristics | open-source, self-developed | Anti-evasion techniques, Robust Tracing | ✓ | CM MP APT | IS organ. | Ranked alerts | Succinct attack graphs | AVIATOR | short |

UNICORN [28] is the first Machine Learning (ML)-based anomaly detector via runtime provenance analytics. It uses a sketch-based, time-weighted provenance encoding scheme to summarize provenance graphs for training and detection. However, unlike aforementioned PIDS, UNICORN does not assist attack investigation by producing attack graphs. Rather, it classifies an entire provenance graph as benign or malicious. UNICORN is evaluated on StreamSpot [116] and DARPA [69] datasets. A major drawback of UNICORN's ML-based approach is the need for representative training data, which are not always available.

MORSE [29] improves on SLEUTH [22] by introducing two key techniques called tag attenuation and tag decay, which help to combat the dependency explosion problem [110], [111] for long-running attacks. Note that SLEUTH [22] can create succinct attack graphs only for short-lived attacks. For long-running attacks, SLEUTH would produce dense attack graphs containing many benign system activities. CONAN [30] proposes a state-based detection framework, in which each system entity, e.g., process and file, is represented as an finite state automata (FSA)-like structure. Combined with suspicious code execution tracking, system entities are assigned with benign or malicious states. As per the authors, state transitions provide another level of context, facilitating efficient APT detection.

TRACE [31] presents a network connection-based cross-machine tracing approach, and proposes a technique called unit-based selective instrumentation to reduce dependence explosion during tracing. TRACE's cross-machine tracing is, however, restricted to system activities of individual instrumented programs, rather than an entire machine's system activities. In addition, such invasive program instrumentation is often too costly to be adopted in practice [27]. Although TRACE's target environment is enterprise networks, it is evaluated only on DARPA-E3 [69] and DARPA-E5 datasets [70], which contain simplified APT attack scenarios not in an enterprise setting, as further discussed in Chapter 6. Besides, TRACE's design is oblivious to the domain context of an enterprise environment. We show in Chapter 8 that TRACE's design is insufficient for combating cross-machine dependency explosion.

ATLAS [32] is a ML-based PIDS that leverages sequence-based model learning for APT attack graph reconstruction. It is applied to detect attacks on individual hosts or small networks consisting of only two hosts. SHADEWATCHER [33] is another ML-based PIDS, which combines recommendation principles with provenance analysis. Like UNICORN, SHADEWATCHER does

not support security analysts in attack investigation. DISTDET [34] is an anomaly-based PIDS focusing on cost-effective deployment of PIDS in practice. It introduces alarm deduplication & semantic aggregation techniques to filter false alarms.

KAIROS [35] is an anomaly-based PIDS that adopts a Graph Neural Network (GNN)-based encoder-decoder architecture for quantifying the degree of anomaly of each system event. It improves on UNICORN, and performs attack detection at a finer granularity. That is, instead of taking an entire provenance graph as input, it takes system events as input, splits the entire time line into many time windows, and classifies each time window as benign or malicious. Yet, this approach produces comparatively dense attack graphs mixing malicious and benign system activities. FLASH [36], as another GNN-based PIDS, performs classification at an even finer granularity level, i.e., the node level. In addition, FLASH presents an embedding recycling database to store the node embeddings from the training phase for improved computational efficiency. Nevertheless, attack graphs produced by FLASH still contain a high percentage of benign system activities, complicating attack investigation. Further ML-based PIDS, e.g., APT-KGL [117], WATSON [118], THREATRACE [114], DEPIMPACT [119], PROGRAPHER [81], MAGIC [115], are not discussed, as they all, like the state-of-the-art ML-based PIDS KAIROS and FLASH, have a limited attack scope: simplified APT attacks.

We argue that, prior PIDS, both anomaly-based [24], [28], [32]–[36] and heuristics-based ones [22], [23], [25]–[27], [29]–[31], would fail at detecting not only multi-phase APT attacks but also cross-machine APT attacks. As it will be further discussed in Chapter 4 and 7, prior PIDS fall short of detecting persistence, due to failure to recognize that persistence attack is always a two-part act. By leveraging persistence, APT attackers break down an entire attack chain into multiple phases, which in turn makes these PIDS create rather disintegrated, seemingly unrelated attack graphs. Besides, prior PIDS are, by construction, limited to intra-machine tracing, and unable to causally associate an attacker's activities across machines in an organization's network(s), as a result of an obliviousness to the network context. This is further discussed in Chapter 4 and 8 in detail.

Unlike these systems, our system CPD presented in Chapter 7 excels in detecting persistence techniques by combining advanced detection rules with pseudo-edges and expert-guided edges. This unique approach enables CPD to effectively identify and investigate persistence threats, bridging the gaps that

other PIDS fail to address. Our system HADES presented in Chapter 8 is the first PIDS capable of performing truly causality-based cross-machine tracing, capitalizing on a critical yet previously undiscovered information: logon session ID. However, we find that HADES is vulnerable to several evasion attack techniques, including persistence. By integrating several preliminary detectors, targeting each evasion attack technique individually, with HADES, our system COMMANDER's whole network tracing achieves a considerably higher level of robustness. Moreover, COMMANDER, presented in Chapter 9, is the first PIDS that incorporates a module for provenance tracing to industrial controllers.

Like HOLMES [25] and RapSheet [26], our systems leverage the synergy of detection rules and heuristics for APT detection and investigation. These detection rules match indicative system activities, and are used to initiate the tracing process or calculate a threat score, which in turn serves to discover related indicative system activities, or rank an attack graph, respectively. But, unlike RapSheet [26], we do not rely on any commercial security product. Comparing to self-developed detection rules in HOLMES [25], self-developed rules in our systems are readily integrable into open-source rule repositories.

We consider that a system offers tracing optimization, if specific techniques are proposed to improve tracing efficiency, completeness, or correctness. While prior systems [22], [24], [26], [27], [29], [31] optimize the tracing process all by enhancing the efficiency, e.g., through optimized path selection or instrumentation, our systems CPD and HADES optimize the tracing process by improving efficiency and completeness, and COMMANDER also improves tracing correctness. It is worth noting that, while none of the presented key techniques in the compared studies is designed for detection and investigation of multi-phase or cross-machine APT attacks, some are complementary to our key techniques for efficient and accurate APT detection and investigation. For instance, tag-based techniques from SLEUTH [22] and MORSE [29], and log unification techniques from OmegaLog [27] could be integrated in our systems for further reducing dependency explosion during tracing.

## 3.2 Threat Model

Like prior provenance-based intrusion detection systems [22]–[36], our systems assume the integrity of the underlying operating systems, firmware and our logging frameworks. Unlike [24]–[26], [28], we do not presume that attacks end before an OS reboot. Our systems are unique in linking provenance attack graphs across reboots using *pseudo-edges* introduced in Chapter 7. Our logging, as a Windows service or a `Systemd` service on Linux, restarts post-reboot, but an OS reboot may miss system audit events. Notably, attack-related processes starting before our logging service are not logged, which often indicates persistence attacks. On Windows, persistence can be achieved through "Create or Modify System Process: Windows Service" (T1543.003 [120]), and on Linux through "Create or Modify System Process: Systemd Service" (T1543.002 [121]). We can trace back to the root process in logs despite missing initial process creation events, using parent process GUID/ID. Our tools System Monitor [84] on Windows and Auditd [85] on Linux record these ID. In addition, our systems focus on detecting long-running attacks in which threat actors already managed to get an initial access to a domain-joined host via (spear-)phishing or program exploitation, and move laterally to other machines leveraging stolen domain credentials instead of program exploitation.

# 4 Problem Analysis

This chapter provides an analysis of the major problems addressed in this thesis. These problems are identified during our examination of existing detection and investigation systems for APT attacks, particularly PIDS, and extensive analysis of threat reports linked in the MITRE ATT&CK knowledge base [48] and posted in leading security vendors' websites, like CrowdStrike [37], Mandiant [52], Microsoft [53], Dragos [54], Trellix [55]. The four major problems of existing systems discussed below answer the research questions ***RQ1.1***, ***RQ2.1***, ***RQ3.1***, and ***RQ4.1*** introduced in Section 1.2, respectively.

## 4.1 Problem One: Lack of attack authenticity in existing APT datasets' attack scenarios

The development of effective APT detection and investigation systems relies heavily on the availability of sound and authentic datasets. To address the

This chapter is based on our previous papers in the following. Note that, to differentiate our own works from others, we cite our own works with a leading letter "P". Inside the chapter, these kind of citations are used in particular for tables and figures taken or adopted from our previous papers:

[P7] Q. Liu, K. Bao, and V. Hagenmeyer, "AVIATOR: A MITRE emulation plan-derived living dataset for Advanced Persistent Threat detection and investigation", in *Proceedings of 2024 IEEE International Conference on Big Data*, 2024, in print

[P4] Q. Liu, M. Shoaib, M. U. Rehman, K. Bao, V. Hagenmeyer, and W. U. Hassan, *Accurate and scalable detection and investigation of cyber persistence threats*, 2024. arXiv: 2407.18832 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2407.18832, under review

[P5] Q. Liu, K. Bao, W. U. Hassan, and V. Hagenmeyer, *HADES: Detecting Active Directory attacks via whole network provenance analytics*, 2024. arXiv: 2407.18858 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2407.18858, under review

[P6] Q. Liu, K. Bao, and V. Hagenmeyer, "COMMANDER: A robust cross-machine multi-phase Advanced Persistent Threat detector", under review

need for such datasets in APT research, the DARPA agency within the US Department of Defense has released several datasets in recent years, including the the DARPA-E3 [69], DARPA-E5 [70] and DARPA-OpTC [71] datasets. These resources have catalyzed the creation of an increasing number of APT detection and investigation systems [18], [28], [29], [33]–[36], [81], [115]. In addition to the DARPA datasets, a separate collection of datasets [32], [34], [122]–[125] have also been made available recently.

Upon reviewing the available datasets, we have unexpectedly discovered a significant disparity between the attack scenarios represented in these datasets and the actual APT attacks that have occurred in the past. Firstly, it is noted that over two-thirds of these datasets are derived from attack scenarios that do not reassemble a complete APT attack chain. Secondly, more than two-thirds of the scenarios lack any persistence attack techniques. Thirdly, the presence of a defense evasion attack stage is observed in less than one-third of the datasets. Fourthly, the initial two DARPA datasets, namely the DARPA-E3 dataset and the DARPA-E5 dataset, fail to encompass a cross-machine or lateral movement attack stage, despite its prevalence in real-world APT incidents. Finally, among the datasets examined, the DARPA-OpTC dataset is the only one in whose attack scenarios the hosts are joined in a domain, reflecting a more realistic enterprise environment. A detailed comparison of these datasets is given in Chapter 6.

## 4.2 Problem Two: Existing PIDS' inability to trace malicious system activities across multiple sessions due to persistence techniques

The SolarWinds attack in 2020, regarded as one of the biggest cyber-attacks ever, has shocked the world and rocked the security industry itself [53], [126]. The most critical part attributed to its success is likely the ability to persist in the victim's environments. The malware later found to be the first involved in this attack is called SUNSPOT, which was used to insert the notorious SUNBURST backdoor into software builds of SolarWinds IT products [127]. The threat actors behind SUNSPOT maintained the persistence by creating a scheduled task set to execute after every machine reboot, a typical persistence technique introduced in the MITRE ATT&CK Matrix [48].

APT actors are progressively redirecting their strategic focus from conventional malware to more nuanced persistence techniques. As defined by MITRE [48], persistence techniques refer to the strategies employed by adversaries to maintain access to systems despite restarts, credential changes, or other disruptions that may sever their connection. These methods generally involve the installation of malicious software or the alteration of legitimate scripts and tasks to guarantee ongoing, unauthorized remote access. Commonly utilized techniques for establishing and sustaining remote connections include reverse shells, SSH, PowerShell Remoting, and various executables. It is noteworthy that persistence techniques were integral to approximately 75% of cyberattacks in 2022 [37].

Another real-world example is the Sandworm APT group's use of webshell persistence in multistage attacks [128]–[130]. APT attackers often break an entire attack chain into phases with waiting periods to avoid detection. They gain initial access, establish persistence, disconnect to evade detection, and later reconnect for further malicious activities. This segmentation and strategic pausing are trademarks of the most stealthy APT attacks, as illustrated in Figure 4.1. We call the time span from the start of a reconnection to the end of this reconnection a phase, and propose the notion of *multi-phase* APT attacks.

In order to motivate the development of CPD, we first show the prevalence of persistence techniques in APT attacks, and then explain why persistence is often misunderstood in academic research, and why a new detection approach is needed.

### 4.2.1 Persistence Prevalence

We extracted data from MITRE ATT&CK knowledge base [131], which encompasses adversary tactics and techniques derived from real-world observations. Our statistical analysis of MITRE's database provides insights into the prevalence of persistence (sub-)techniques in real world. There are 99[1] distinct (sub-)techniques associated with the persistence tactic. Our findings indicate

[1] If a technique has at least one sub-technique, only its sub-techniques are counted. Otherwise, it is redundant.

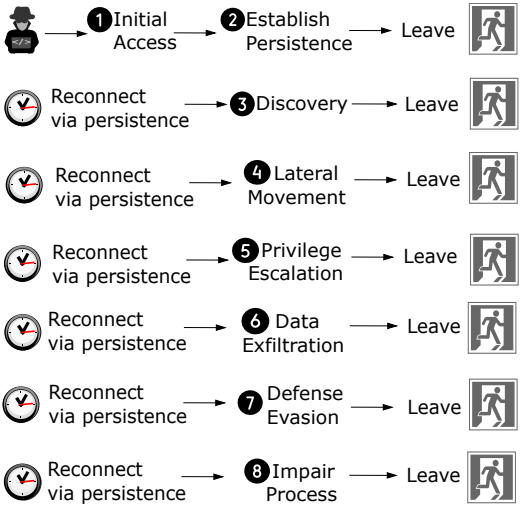



**Figure 4.1:** Stealthiness by persistence [P4].

that 94 out of 136 APT groups (69%) have utilized at least one persistence (sub-)technique in their activities.

We rank both the persistence (sub-)techniques according to the number of APT groups that have employed these techniques in practice, as well as the APT groups based on the number of persistence (sub-)techniques they have utilized. The top 10 persistence (sub-)techniques are presented in Table 4.1, while the top 10 "persistent" APT groups are detailed in Table 4.2. As the Table 4.1 shows, the most popular persistence sub-technique used by APT groups is the "Registry Run Keys / Startup Folder", referred as T1547.001 [132] in the MITRE ATT&CK Matrix and leveraged by more than one-third of all APT groups. Thus, this persistence sub-technique is taken as an example throughout the Chapter 7, where we introduce our dedicated persistence detector Cpd. Further, Table 4.2 reveals that APT29 is the most "persistent" APT group, whose use of persistence techniques in the past cover a quarter of all persistence techniques.

**Table 4.1:** Top 10 persistent (sub-)techniques [P4].

| Registry Run Keys / Startup Folder | Scheduled Task | Web Shell | DLL Side-Loading | External Remote Services | Windows Service | Domain Accounts | WMI Event Subscription | DLL Search Order Hijacking | Local Account |
|---|---|---|---|---|---|---|---|---|---|
| 49 | 45 | 23 | 21 | 20 | 20 | 11 | 10 | 9 | 9 |

**Table 4.2:** Top 10 persistent APT groups [P4].

| APT29 | APT41 | Lazarus Group | Sandworm Team | Kimsuky | APT28 | APT39 | APT3 | Magic Hound | Threat Group-3390 |
|---|---|---|---|---|---|---|---|---|---|
| 25 | 16 | 12 | 11 | 10 | 10 | 8 | 8 | 8 | 8 |

### 4.2.2 Persistence Misconception

Persistence is frequently observed in real-world APT attacks; however, it is often misinterpreted in academic studies. According to MITRE, persistence refers to the capability of maintaining access to compromised systems despite restarts, credential changes, and other disruptions that may temporarily sever access. This can be accomplished primarily in two ways: attackers may either establish a remote connection using compromised credentials, e.g., T1078 [133], or insert a public key into the `SSH authorized_keys` file, i.e., T1098.004 [134]. Alternatively, attackers can initiate a connection from within the victim system by adding a new entry to the Registry Run keys or startup folder (T1547.001 [132]), creating scheduled tasks or jobs (T1053 [135]), or embedding commands in Unix shell configuration files such as .bashrc (T1546.004 [136]). For additional information on these techniques, we direct the reader to the MITRE ATT&CK Matrix [20].

In our examination of various public APT datasets [32], [70], [71] for the assessment of PIDS, we discovered a notable deficiency in the comprehension of persistence. Effective persistence relies on three essential criteria: setting up a trigger (such as creating a Registry Run key), associating code for remote access with this trigger (for instance, placing an executable in the Registry Run key), and successfully initiating a remote connection upon the trigger's activation. In practice, the third criterion may not always be fulfilled, leading

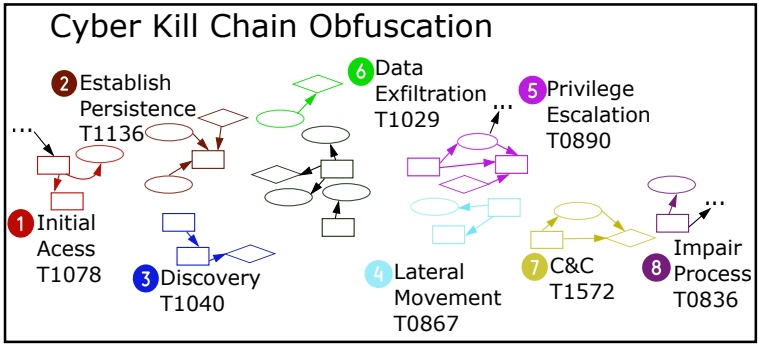



**Figure 4.2:** Obfuscation of APT attack stages in a provenance graph.

APT groups, including APT29, to implement multiple persistence techniques to improve their chances of remaining in the target environment. Unfortunately, current datasets typically fulfill only the first criterion by setting up a trigger, often linking it to an irrelevant value, thus neglecting the second and third criteria. This oversimplification can be detrimental, as legitimate programs frequently activate these persistence triggers, allowing attackers to blend seamlessly into normal system operations. In contrast, each of MITRE's eleven emulation plans, grounded in actual APT behaviors, includes at least two persistence techniques without such oversimplification.

### 4.2.3 Consequence for Existing PIDS

Anomaly-based or learning-based PIDS [28], [33], [35], [36], [81], [114], [115], [137] aim to identify novel attacks with minimal prior knowledge. These systems model benign behavior through provenance graphs and detect deviations from the established normal behavior. However, the premise that attack-related activities are inherently anomalous does not always hold true. Our assessment in Section 7.4 shows that state-of-the-art learning-based PIDS are not only slower but also exhibit reduced accuracy in detecting persistence, primarily due to their inability to learn persistence attacks' semantics.

Similarly, prior heuristic-based PIDS also face challenges in understanding the semantics of persistence attacks, which complicates the process of linking fragmented stages of APT attacks. This often leads to incomplete and disjointed reconstructions of provenance (attack) graphs. For instance, for

the attack scenario illustrated in Figure 4.1, heuristic-based PIDS, such as HOLMES [25] and RapSheet [26], often fail to piece together the full scope of an attack involving persistence. Instead, they generate isolated graphs that represent only segments of the APT attack chain. It is important to note that when persistence techniques are employed, TTP (Tactics, Techniques, Procedures) security alerts for each stage of the attack become unlinked, in contrast to the Figure 2.5 introduced in Chapter 2.

These systems prioritize attack graphs based on the number of APT stages, such as privilege escalation and data exfiltration, present within the graphs. If an attacker successfully fragments the attack graph into smaller, disconnected segments, each segment is inherently assigned a lower severity score, and is subsequently ranked lower for detection and investigation.

## 4.3 Problem Three: Existing PIDS' inability to trace malicious system activities across machines

In order to motivate the development of HADES, we first describe the prevalence of cross-machine attack stages in real-world APT attacks, and then explain why Active Directory is most misused for this purpose. Most importantly, we illustrate what kind of problem cross-machine attacks can cause for existing PIDS, particularly in terms of tracing.

### 4.3.1 Prevalence of cross-machine attack stages

In real-world APT attacks, attackers generally gain initial access through a less secure machine and subsequently seek to move laterally to other systems that contain critical data of the target organization. Following this initial access, there is a marked increase in the focus on Microsoft Active Directory, as attackers aim to gather intelligence about the internal network and acquire essential domain credentials, thereby facilitating lateral movement within the victim's network. For example, the frequency of Kerberoasting attacks [138] surged by nearly 600% from 2022 to 2023, while Pass-the-Hash attacks [139] saw a 200% increase. The exploitation of compromised domain credentials has emerged as the predominant method for attackers to traverse a target network, accounting for 80% of contemporary cyberattacks. Furthermore,

half of the organizations globally have reported experiencing such attacks in recent years, with 40% of these incidents proving to be successful [40], [42].

### 4.3.2 Why Active Directory is misused for cross-machine attack stages?

Active Directory (Domain Service) is essential for identity and access management (IAM) within the IT infrastructures of modern enterprises, with 90% of Fortune 1000 companies depending on it [40], [45]. Active Directory (AD) utilizes a database to maintain vital information regarding an organization's network resources, enabling administrators to control access to these resources effectively. In essence, Active Directory holds the blueprint of an organization's environment, and the keys to all available resources. Following initial access, attackers frequently exploit Active Directory functionalities at various stages of the Cyber Kill Chain, particularly during internal reconnaissance, credential access, lateral movement, and privilege escalation. As a result, numerous attacks targeting vulnerabilities in Microsoft's Active Directory design and implementation have emerged over time, leading to partial or even complete compromises of many enterprise networks [140], [141].

### 4.3.3 Consequence for Existing PIDS

Figure 4.3 shows two naive cross-machine tracing approaches, which may be employed in existing PIDS. Network connection-based tracing, which naively connects two intra-machine provenance graphs whenever there is a network connection between the two machines, as shown in Figure 4.3 (a), would lead to a dense graph indiscriminately connecting all system activities as depicted in Figure 4.4 (a). Logon-based tracing in Figure 4.3 (b) reduces false dependencies by considering only logon-initiated network connections, resulting in less dense graphs in Figure 4.4 (b). However, in the prevalence of identity theft, it is infeasible to discern the true identity behind each logon. That is, a provenance graph produced from logon-based tracing can still contain system activities of multiple identities. We consider this as a cross-machine dependency explosion problem, as, in this case, an attack graph produced by a PIDS would contain system activities of benign users,

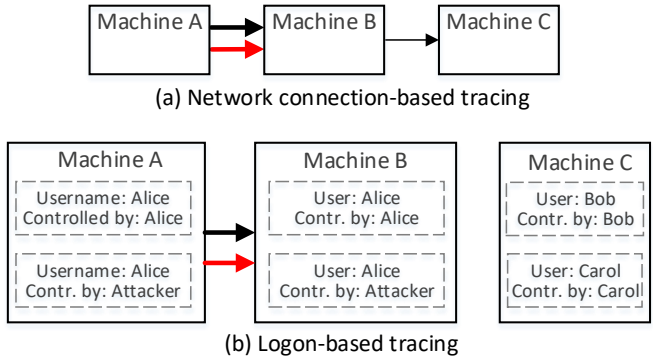



**Figure 4.3:** Naive cross-machine tracing approaches [P6]. Whereas a thin black edge denotes a network connection, a thick black edge denotes a logon by a genuine user and a thick red edge denotes a logon by an attacker.

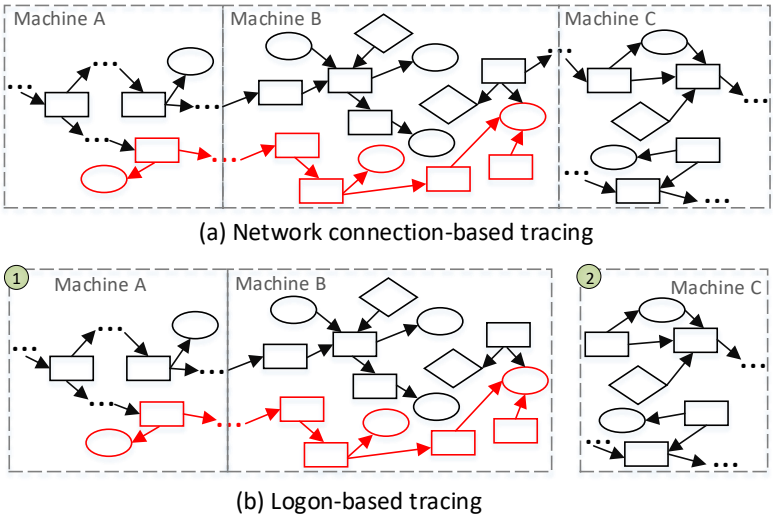



**Figure 4.4:** Results from naive cross-machine tracing approaches [P6]. Whereas black denotes system activities originated from genuine users, red denotes system activities caused by attackers. If a tracing approach returns more than one graph, these graphs are numbered and considered as independent for further investigation.

and these benign system activities may be falsely hold responsible for truly malicious system activities. Whereas intra-machine dependency explosion problem occurs for long-running processes [110], [111], in which each input is conservatively considered causally responsible for all subsequent outputs, and vice versa, cross-machine dependency explosion happens if cross-machine edges are created merely on a network connection basis.

To enhance clarity and ease of understanding, we assume that each machine depicted in Figure 4.3 has only two logon sessions. Additionally, it is important to highlight that we use different colors to distinguish between benign and malicious system activities solely for illustrative purposes. The primary objective of precise tracing is to associate system activities with the same identity, whether that identity is an attacker or a legitimate user, while also differentiating activities that belong to distinct identities. Therefore, the color coding serves merely as a visual aid for the readers, and is transparent to the tracing.

The primary goal of tracing is to distinctly differentiate system activities across various identities, rather than categorizing these activities as either benign or malicious. However, the detection accuracy of PIDS, as demonstrated in studies such as [25], [26], is contingent upon the comprehensiveness of the generated attack graphs. This comprehensiveness is fundamentally linked to the reliability of the tracing methodology employed. In the final step, PIDS typically prioritize attack graphs based on their threat scores, which are determined by the indicative attack steps or techniques identified within the respective graphs. Consequently, an incomplete attack graph may result in a diminished threat score, potentially causing a true attack graph to be assigned a lower rank than that of false-positive attack graphs.

## 4.4 Problem Four: Existing PIDS' inability to robustly trace malicious system activities across an organization

The aforementioned **Problem Three** can be solved by HADES introduced in Section 5.3, with a novel concept called *logon session-based execution partitioning and tracing*, which tackles the cross-machine dependency explosion problem. HADES can link system activities belonging to the same identity, and

separate system activities unrelated to each other into individual provenance graphs. However, HADES is based on the assumption that attackers would not conduct evasive attack techniques that can disrupt its tracing process.

Yet, our analysis of APT threat reports linked in the MITRE ATT&CK knowledge base [48] yields the finding that HADES, as the first provenance-based system capable of accurate cross-machine tracing, is susceptible to several evasive attack tactics/techniques routinely employed by APT actors: persistence [142], session hijacking [143], and port forwarding [144]. Whereas port forwarding attacks can directly interrupt HADES's cross-machine tracing, both persistence and session hijacking attacks hinder accurate intra-machine tracing, on which cross-machine tracing hinges. The impacts of these evasion techniques on HADES are discussed below in detail.

### 4.4.1 Evading HADES via Persistence

As it will be further discussed in Section 5.2, an effective persistence is always a two-phase process, comprising a setup phase and an execution phase. The setup phase primarily functions as a preparatory stage, while the execution phase unveils the true intentions of the attackers. In the setup phase, attackers often exploit legitimate system features, such as the Registry Run keys / Startup Folder [132] or Windows Service [120], to deploy code that acts as a Command and Control (C2) agent. Subsequently, this C2 agent is automatically activated during the execution phase. During this execution phase, the C2 program is required to either establish or receive a remote connection with the attacker to obtain further instructions, thereby revealing the true objectives of the attack.

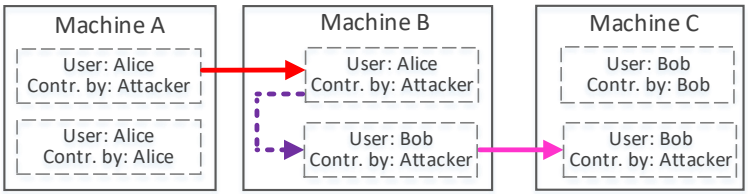
The persistence execution phase typically commences with either a machine reboot or a new user logon, resulting in associated system activities operating under a new logon session. Consequently, the persistence setup phase and the persistence execution phase frequently occur in separate logon sessions, which may be associated with the same user or different users. Due to a lack of awareness regarding persistence attacks, the disconnection between the relevant logon sessions during intra-machine tracing is inevitable. With logon session-based execution partitioning and tracing, the absence of cross-session connections in intra-machine tracing can result in inaccurate and incomplete cross-machine tracing.

An example of how accurate cross-machine tracing can be hindered via persistence is given in Figure 4.5. Figure 4.5 (a) shows a persistence attack inside the Machine B, where the persistence setup is conducted in a logon session under user Alice, and the persistence execution is run in a logon session under user Bob. This is possible when the attacker drops a C2 agent program in the Machine B and links the path to this program with the Registry Run key under user Bob. Therefore every time user Bob logs in, the C2 program is automatically started by the OS and connects back to the same attacker for further commands. Note that the logon session associated with persistence execution, i.e., the second session in the Machine B, can be under the user Alice as well, if the attacker links the path to the C2 program with the Registry Run key under user Alice, or even under the System user, if, for instance, the attacker hides the C2 program in a Windows service.

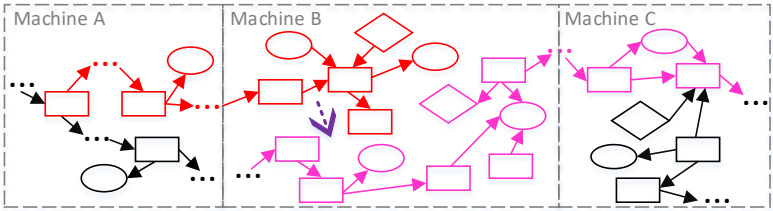
The link between persistence setup and persistence execution cannot be established by parsing system logs alone. Only a persistence detection system like CPD that accurately detects persistence setups and persistence executions of various persistence attack techniques curated by MITRE [48] can discover the broken links and rebuild them into the provenance graphs. As HADES is unaware of evasion attack via persistence, the dashed violet edge indicating a persistence attack in Figure 4.5 (b) is invisible, leading HADES to attribute the system activities in red and the system activities in pink to two different identities in Figure 4.5 (c), although they are conducted by the same attacker.

### 4.4.2 Evading HADES via Session Hijacking

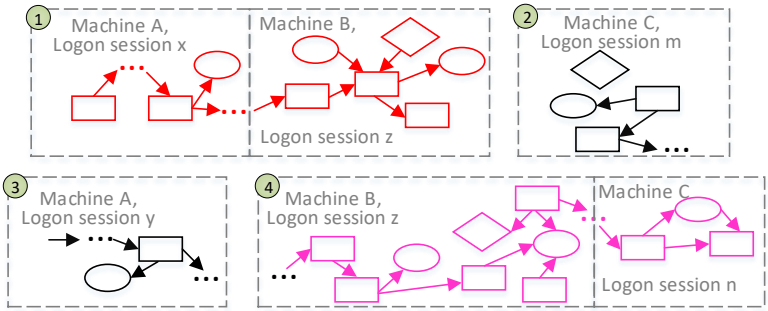
In this thesis, as session hijacking, we refer to console / remote desktop session hijacking [143], [145], or more precisely logon session hijacking, rather than browser/web session hijacking. Note that console / remote desktop session and logon session are two different concepts, and are at different abstraction levels. A console / remote desktop session contains at least one logon session, and a console / remote desktop session hijacking indicates hijacking of all logon sessions within it. With sufficient privilege on one session, attackers can hijack another existing session. A typical reason for an attacker to conduct session hijacking is that the hijacked session may contain more (credentials-related) information enabling the attacker to further access other devices in the network.




(a) Evasion of logon session-based tracing inside one machine




(b) Provenance graph before logon session-based tracing




(c) Provenance graphs after logon session-based tracing

**Figure 4.5:** Evasion of cross-machine tracing via persistence or session hijacking [P6]. Whereas black denotes benign system activities, red, violet and pink all denote malicious system activities. The dashed violet edges indicate a persistence or session hijacking attack.

Like persistence attacks, a session hijacking attack is not directly evident in system logs, meaning that the link between a hijacking logon session and a hijacked logon session cannot be created by solely processing system logs. If a session hijacking attack is not detected, the hijacking session would not be linked to the hijacked session during logon session-based tracing, inevitably

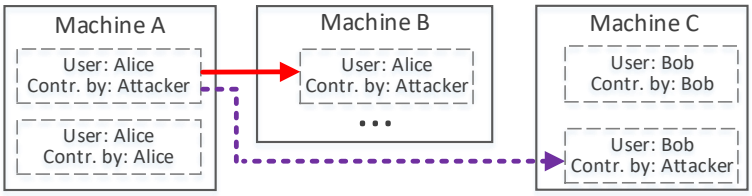
leading to premature end of whole network tracing. The dashed violet edge in Figure 4.5 (a) and (b) can also indicate a session hijacking attack, apart from a persistence attack. Unlike in a persistence attack, the second session, i.e., the hijacked session, in the Machine B is typically under a user different from the user in the first session, i.e., the hijacking session. Being oblivious to evasive session hijacking attacks, and hence to the dashed violet edge in Figure 4.5 (b), HADES will attribute the system activities in red and the system activities in pink to two different identities, as shown in Figure 4.5 (c), even though they are in fact caused by the same attacker.

### 4.4.3 Evading HADES via Port Forwarding

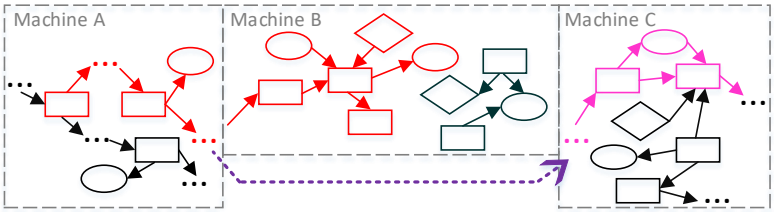
Network segmentation often compels attackers to utilize port forwarding to access further machines through an already compromised machine, also known as proxy [144]. When port forwarding is utilized, the source IP address recorded in a logon event reflects the IP address of the proxy machine rather than that of the actual source machine[2]. Accurate identification of the source IP address is essential for HADES's cross-machine tracing. During this process, HADES extracts the source IP address from each logon event and cross-references it with authentication events gathered from the domain controller and system events on the source host (identified through the IP address) to verify that an authentication request originated from this host and that a logon was indeed initiated from it. If both the authentication request and logon initiation are validated, HADES will proceed to analyze subsequent events to determine the logon session and user name, among other details, and continue the tracing process. Conversely, if these confirmations are not met, HADES will halt further tracing from that logon event.

Apparently, taking the IP address of the proxy machine would lead to premature termination of the tracing, as the logon was not initiated from the proxy machine. Figure 4.6 (a) shows a port forwarding attack scenario, in which the attacker from the Machine A has logged on the Machine C via port forwarding through the Machine B. Being oblivious to port forwarding, HADES is unable to create the dashed violet edge in Figure 4.6 (b), and end up

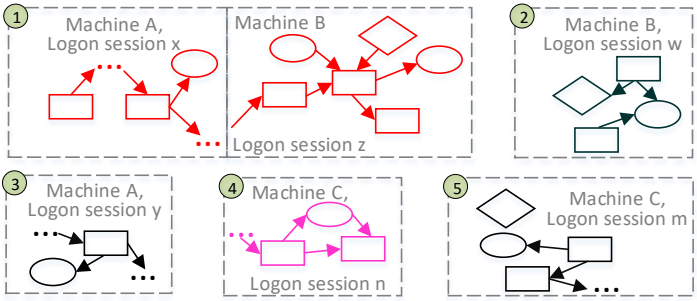
[2] It is important to note that most logon events do not include the source host name. The source host name is only present when NTLM (New Technology LAN Manager) is selected as the authentication method, as opposed to the default Kerberos authentication process.




(a) Evasion of logon session-based tracing between two machines




(b) Provenance graph before logon session-based tracing




(c) Provenance graphs after logon session-based tracing

**Figure 4.6:** Evasion of cross-machine tracing via port forwarding [P6]. The dashed violet edges indicate a port forwarding attack step.

wrongly attributing the system activities in red and in pink to two different identities, as shown in Figure 4.6 (c). That is, naively relying on the IP address in a logon event could lead to incomplete cross-machine tracing.

# 5 Approach Overview

This chapter presents an overview of our approach. Although the problems discussed in Chapter 4 all exhibit in current research on APT detection and investigation, these problems per se are distinct from each other, and can be first addressed separately. Hence, through the development of this thesis, we adopt a strategy, in which we successively and progressively solve these problems. That is, to address the **Problem One** introduced in Chapter 4, we present AVIATOR, which is discussed in the Part One (Section 5.1) of our approach overview. Further, we develop CPD independently targeting the **Problem Two**, and describe it in the Part Two (Section 5.2) shortly. Then, we create HADES to specifically tackle the **Problem Three**, and briefly discuss it in the Part Three (Section 5.3). On top of that, we build our final system COMMANDER in the Part Four (Section 5.4), which relies on the presence of the first three systems and handles the **Problem Four**. COMMANDER is considered as the final product of this thesis, which unifies the contributions from each of our systems, and provides a solution for all above problems.

---

This chapter is based on our previous papers in the following. Note that, to differentiate our own works from others, we cite our own works with a leading letter "P". Inside the chapter, these kind of citations are used in particular for figures taken or adopted from our previous papers:

[P7] Q. Liu, K. Bao, and V. Hagenmeyer, "AVIATOR: A MITRE emulation plan-derived living dataset for Advanced Persistent Threat detection and investigation", in *Proceedings of 2024 IEEE International Conference on Big Data*, 2024, in print

[P4] Q. Liu, M. Shoaib, M. U. Rehman, K. Bao, V. Hagenmeyer, and W. U. Hassan, *Accurate and scalable detection and investigation of cyber persistence threats*, 2024. arXiv: 2407.18832 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2407.18832, under review

[P5] Q. Liu, K. Bao, W. U. Hassan, and V. Hagenmeyer, *HADES: Detecting Active Directory attacks via whole network provenance analytics*, 2024. arXiv: 2407.18858 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2407.18858, under review

[P6] Q. Liu, K. Bao, and V. Hagenmeyer, "COMMANDER: A robust cross-machine multi-phase Advanced Persistent Threat detector", under review

Figure 5.1 presents our approach overview, in which we color-code the building blocks to attribute their origins to each of our systems. Only the building block in grey does not originate from our systems, but from Sigma [74] and Elastic [146], which are two popular open-source detection rule repositories. The building blocks in green derive from our system AVIATOR, and provide the main evaluation dataset for our other systems. AVIATOR includes both system logs and authentication & logon logs from all machines in the emulation infrastructure.

The building blocks in light blue stem from our system CPD, which can operate either standalone as a persistence detector, or as an addon for COMMANDER, in which it provides COMMANDER with critical information for robust tracing. Yellow and light-yellow building blocks come from our system HADES, whose main purpose is accurate cross-machine tracing. Orange building blocks are proposed in our system COMMANDER; they serve to, on the one hand, enhance the tracing robustness, and on the other hand, rank the attack alerts/graphs. The building blocks in purple include techniques that are presented in COMMANDER, but are actually adapted from HADES.

All the building blocks in Figure 5.1 are further discussed in below sections and/or in Chapter 6 - 9, respectively, in detail.

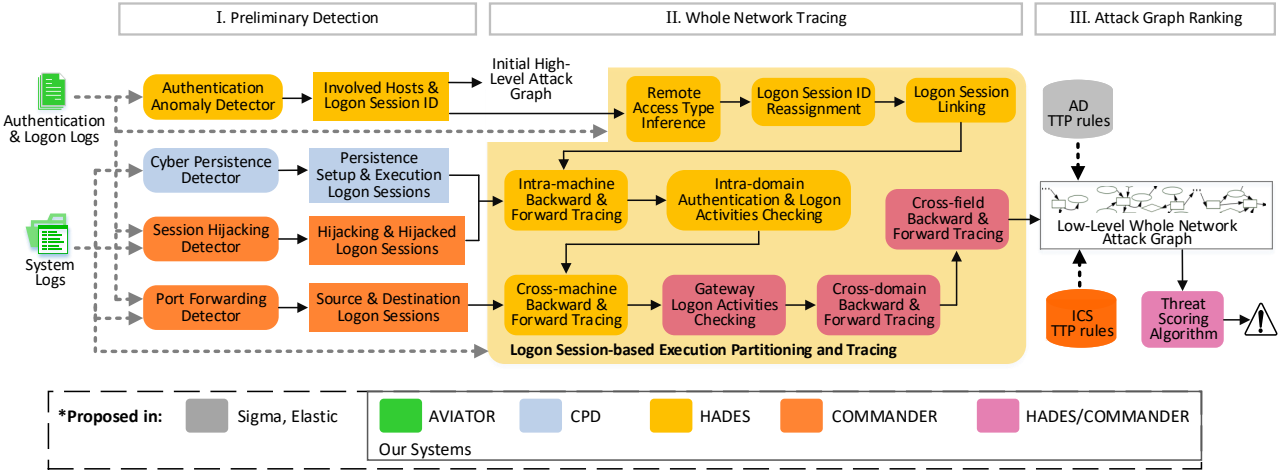



**Figure 5.1:** Approach overview [P6].

## 5.1 Part One: AVIATOR

Recognizing the weaknesses of existing datasets, we create AVIATOR (**A** MITRE emulation plan-derived li**VI**ng d**AT**aset f**OR** APT detection and investigation)[1] to stimulate further research. Our dataset AVIATOR is created by implementing MITRE emulation plans [46]. MITRE, as a prominent organization within the security sector, has released eleven comprehensive emulation plans derived from the analysis of historical APT attack campaigns. Out of these eleven plans, nine feature detailed attack scenarios where the hosts within the targeted enterprise network are appropriately integrated into a domain through Microsoft Active Directory, allowing for multiple cross-machine attack stages that mirror actual APT incidents. Each emulation plan encompasses various stages focused on persistence and defensive evasion, thereby enhancing the authenticity of the attacks. Furthermore, these emulation plans are employed in MITRE Engenuity ATT&CK® Evaluations [147] to evaluate commercial security solutions from leading vendors in the industry.

Unfortunately, MITRE has not made any corresponding datasets available; the emulation plans are solely intended as a red team [148] tool, with auditing and logging considerations excluded from their scope. For example, in MITRE Engenuity ATT&CK® Evaluations [147], it is the responsibility of detection system developers to establish and configure their own monitoring infrastructure. Consequently, we resort to implementing MITRE's emulation plans following the guidelines provided by MITRE to create our dataset AVIATOR.

In comparison to existing datasets for research on APT detection and investigation, our dataset AVIATOR exhibits the highest complexity and authenticity in attack scenarios, as elaborated in Chapter 6. Furthermore, AVIATOR has been designed with a focus on operability, usability, reproducibility, and extensibility, for which current datasets fall short. In addition to the AVIATOR dataset, we offer log shipping tools, log parsing tools, and logging configuration files to facilitate other researchers in creating their own datasets, which may be more suitable for evaluating their detection systems. Additionally, we plan to incorporate more log types in future iterations of our dataset AVIATOR.

[1] https://gitlab.kit.edu/kit/iai/rsa/aviator

## 5.2 Part Two: CPD

To tackle the challenges associated with current persistence threat detection, we present Cyber Persistence Detector (CPD), a novel system designed for rapid and precise identification of persistence threats. Our approach refrains from making overly optimistic assumptions regarding persistence behavior, minimizes both false positives and false negatives, triages persistence-related threat alerts, and produces accurate attack graphs to facilitate swift incident response.

CPD is founded on an extensive examination of the MITRE ATT&CK framework [20], which is acknowledged as the most comprehensive and widely utilized repository of persistence threats. Our analysis revealed that successful persistence attacks typically unfold in two distinct phases: the persistence setup (for instance, the creation of a Registry Run key) and the persistence execution (such as a remote connection initiated by that key). *The persistence setup is merely preparatory, while the persistence execution reveals the attackers' true motives.* Understanding this two-phase process is essential, yet it is neglected by current detection systems.

Utilizing the aforementioned key insight, CPD presents a novel concept of *pseudo-dependency edges* (pseudo-edges) to link persistence setup and execution activities within system logs, thereby forming a comprehensive provenance graph. CPD starts by recording every persistence setup activity in system logs, and then specifically checks if there is a subsequent remote connection, i.e., potential persistence execution, that can be traced back to the persistence setup activity. In the absence of such a connection, no alert is generated, which significantly minimizes false positives. Conversely, if a connection is identified, a pseudo-edge is established, and additional contextual indicators, as elaborated in Section 7.2.3, are utilized to determine whether it constitutes a persistence attack. The formation of a pseudo-edge involves tracing processes in provenance graphs that initiate or receive remote connections, assessing their alignment with persistence setup and persistence execution as per our advanced detection rules.

This approach allows CPD to detect potential persistence activities through established rules and to scrutinize processes involved in remote connections, assessing their significance within This approach enables CPD to identify potential persistence activities through detection rules and to analyze processes engaged in remote connections, assessing their role in the broader scheme

of a persistence threat. This dual-layered approach, which combines precise activity tracking with the creation of pseudo-edges, empowers CPD to exceed the capabilities of current threat detection systems.

Relying exclusively on pseudo-edges to enhance the accuracy of persistence detection, despite its benefits, poses certain challenges. A significant issue is that even the most thorough logging systems may fail to capture every connection, which can result in gaps within the provenance graph and lead to false negatives. Our research indicates that the integration of Windows ALPC (Asynchronous Local Inter-Process Communication) logs with system audit logs can help address these gaps. However, this approach considerably increases the requirements for log storage. In contrast, CPD employs a novel technique called *expert-guided edges*, which leverages insights from process creation and system policy to refine the provenance graph while reducing log volume by an average of 37%. While pseudo-edges are particularly useful for identifying persistence threats, expert-guided edges have a wider application, enhancing overall efficiency in tracing and log management beyond merely detecting persistence threats.

Another challenge is the excessive generation of pseudo-edges due to the behaviors of certain benign programs, which can complicate the detection process. To mitigate this issue, CPD implements a false positive reduction algorithm informed by a comprehensive analysis of APT behaviors. This approach ensures that only truly malicious activities are flagged. Our algorithm is based on the crucial understanding that persistence is merely one component of a multi-stage APT kill chain, all elements of which must operate in concert to achieve the attackers' goals. By verifying the presence of related kill chain techniques and tactics in close proximity within the provenance graph, CPD significantly improves its ability to differentiate between benign and malicious actions, thereby enhancing both the accuracy and reliability of threat detection.

Our system CPD outperforms existing detection systems, as evidenced by evaluations on both public datasets and those derived from strictly implemented MITRE emulation plans. It excels in reducing false positives by 93% on average, effectively detecting true persistence attacks, and producing succinct attack graphs that explain the persistence setup and execution. The capability to pinpoint persistence setup and execution within a provenance graph and present it in context provides security analysts with actionable insights for further investigation, demonstrating CPD's practicality.

## 5.3 Part Three: HADES

To overcome the restriction of PIDS' tracing scope, we design HADES, which recognizes the critical importance of logon session ID, and tackles the cross-machine dependency explosion problem with the novel concept logon session-based execution partitioning and tracing. By leveraging both authentication & logon logs and system logs, our system HADES is capable of 1) fine-grained cross-machine provenance tracing, 2) alleviating dependency explosion also for intra-machine provenance tracing, 3) drastically reducing log size, 4) automatically pinpointing privilege escalation.

HADES employs a two-stage approach for efficient detection of AD-based cross-machine APT attacks. Its first stage consists of a lightweight authentication anomaly detector responsible for identifying potential AD attacks and forwarding its results to HADES's stage two component, which performs logon session-based tracing and attack graph triage. Both our authentication anomaly detection model and attack graph triage algorithm are rooted in our thorough analysis of AD attacks.

Through HADES's development, we faced several difficulties in realizing accurate cross-machine tracing. First, depending on remote access type, each authentication & logon process causes varying number of logon events with distinct logon session ID, complicating the identification of the correct session ID. Second, under certain circumstances, system activities in a new logon session are assigned with an existing session ID, causing false dependencies. Third, it is often not possible to disclose the remote access type by examining the current logon event alone. We overcome these challenges by introducing a *remote access type inference* module, a *logon session ID reassignment* module, and a *logon session linking* module in HADES, based on our extensive profiling and analysis of Windows logging frameworks. Our implementation of logon session-based tracing avoids time-consuming instrumentation, error-prone training, and applies to the whole system rather than a single program.

Comparing to naive cross-machine tracing approaches showed in Figure 4.3, logon session-based tracing can narrow down system activities deemed as relevant from an entire machine to just one logon session, as shown in Figure 5.2. That is, it can truly linking system activities belonging to the same identity, and separating system activities unrelated to each other into individual provenance graphs displayed in Figure 5.3.

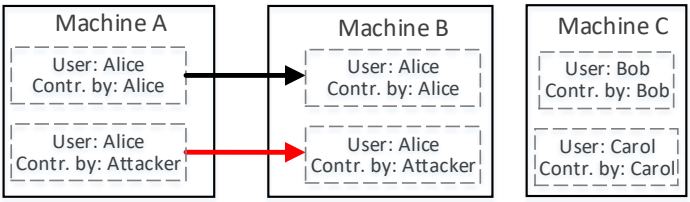



**Figure 5.2:** Logon session-based tracing. Whereas a thin black edge denotes a network connection, a thick black edge denotes a logon by a genuine user and a thick red edge denotes a logon by an attacker.

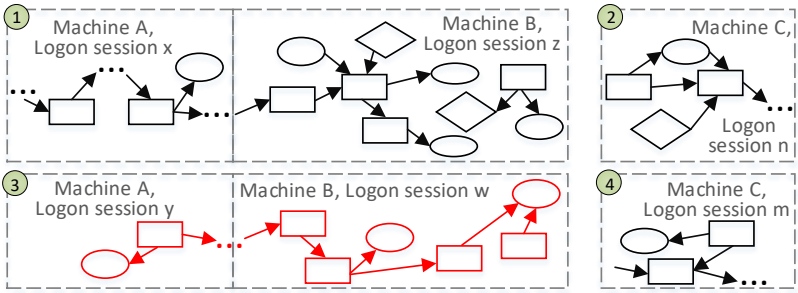



**Figure 5.3:** Result from logon session-based tracing [P6]. Whereas black denotes system activities originated from genuine users, red denotes system activities caused by attackers. If a tracing approach returns more than one graph, these graphs are numbered and considered as independent for further investigation.

## 5.4 Part Four: COMMANDER

Recognizing the weaknesses of HADES described in the **Problem Four** in Chapter 4, we propose COMMANDER (**CrOss-Machine Multi-phase AdvaNced persistent threat DEtectoR**). COMMANDER integrates CPD with HADES, and incorporates another two specialized detectors for session hijacking and port forwarding, respectively. COMMANDER features a three-stage approach for efficient and accurate attack detection and graph reconstruction. The first stage of COMMANDER serves as preliminary detection, and accommodates four specialized detectors running in parallel, i.e., an authentication anomaly detector, a cyber persistence detector, a session hijacking detector and a port forwarding detector. The authentication anomaly detector identifies

anomalies indicating identity-driven attacks, and produces an initial high-level attack graph for each anomaly, which is passed to COMMANDER's second stage: whole network tracing. The other three detectors make complementary contributions to ensuring robust and correct whole network tracing, by delivering critical information to guide and adjust the tracing process.

Robust and causally accurate cross-machine tracing is crucial for accurate and swift detection and investigation of sophisticated APT attacks. The objective of whole network tracing is to distinguish system activities associated with different identities while establishing causal links between the system activities of the same identity across all machines within the organization. Achieving accurate and thorough tracing at this stage is critically important for the ranking of attack graphs in COMMANDER's final stage, where graphs are evaluated based on their threat scores, which are derived from the set of indicative attack steps and security alerts identified within the respective graph. Consequently, any inaccuracies or omissions in tracing may result in true attack graphs being assigned lower rankings than those of false positives.

Further, COMMANDER is designed as a PIDS for detecting cross-machine attacks in industrial-sector organizations consisting of an enterprise network and an industrial control system (ICS), which are typically in two separate domains. That is, COMMANDER is able to trace across domains, given that the enterprise network and the ICS network are often connected via a gateway server. Besides, we develop parsers for system logs from two popular industrial controllers in energy systems, i.e., Siemens SIPROTEC [49] and Hitachi Energy RTU500 [50], and TTP detection rules with a reference to the MITRE ATT&CK Matrix for ICS [51] for these controllers. COMMANDER is capable of accurately attributing the activities on these controllers to the true identity behind those actions, even if the access originates from the enterprise network.

# 6 AVIATOR Dataset

The increasing demand for the development of new detection and investigation systems for Advanced Persistent Threat (APT) has highlighted the pressing issue of insufficient reliable and authentic datasets. In recent years, there has been a rapid release of new datasets aimed at enhancing research on APT detection and investigation. However, our analysis of the available datasets reveals a considerable disparity between the attack scenarios they present and the actual APT attacks that occur in the real world. Acknowledging the shortcomings of previous datasets, particularly regarding the complexity and authenticity of attack scenarios, we create a new, sound dataset named Aviator, which is supported by MITRE emulation plans. MITRE has published nearly a dozen emulation plans that effectively replicate the real-world attack campaigns of APT groups observed in the past. Nevertheless, MITRE has not made any datasets available. Consequently, we have stringently implemented these emulation plans, and expanded them to include an industrial control system and attack steps on it, mimicking APT groups most known for their attacks against critical infrastructures in the past.

## 6.1 Challenges & Objectives

During the development of Aviator, we faced several challenges. Firstly, in each emulation plan, MITRE has only made available instructions and certain

This chapter is based on a previous paper of ours. Note that, to differentiate our own works from others, we cite our own works with a leading letter "P". Inside the chapter, these kind of citations are used in particular for tables, figures and algorithms taken or adopted from our previous paper:

[P7] Q. Liu, K. Bao, and V. Hagenmeyer, "AVIATOR: A MITRE emulation plan-derived living dataset for Advanced Persistent Threat detection and investigation", in *Proceedings of 2024 IEEE International Conference on Big Data*, 2024, in print

resources related to the hacking tools used, without providing its emulation infrastructure, e.g., as exported virtual machine images or snapshots. Consequently, we had to construct our own emulation infrastructure based on the design of MITRE's emulation infrastructure. Secondly, the implementation of certain attack steps is not straightforward, as the instructions do not consist of executable commands but rather function as guidelines. We have observed that the hacking tools and scripts provided do not always operate as intended and require modifications. Lastly, since MITRE does not offer any guidance regarding logging, we must develop the logging infrastructure on our own.

Additionally, we have expanded two MITRE emulation plans, specifically Sandworm [87] and Oilrig [89], to incorporate an industrial control system (ICS) infrastructure along with the associated attack steps on it, as the original MITRE emulation plans do not address attack steps targeting ICS. The APT groups featured in these emulation plans are particularly recognized for their real-world attacks on ICS, including notable incidents involving the Ukrainian power grid [54], [149]–[151] and attacks on the energy and chemical sectors in the Middle East [152]–[154]. The current version of our dataset AVIATOR comprises data derived from the execution of four original and two extended MITRE emulation plans.

The rationales for making AVIATOR a living dataset are threefold. Firstly, we believe that one of the most significant obstacles in creating accurate detection systems is the challenge of determining in advance what data should be logged. In the course of developing our first APT detection and investigation system CPD, which is discussed in Section 7, we initially attempted to rely solely on system logs obtained through System Monitor (Sysmon) [84]. However, we discovered that Sysmon fails to log file and Registry read activities, which resulted in critical gaps in our ability to detect specific attack techniques. Consequently, we were compelled to configure the Windows Security Log [155] to capture additional system logs, necessitating a rerun of the emulation plans to gather the required data. Furthermore, during the development of our second APT detection and investigation system HADES, which is discussed in Section 8, we recognized that relying exclusively on system logs was inadequate for the accurate detection of cross-machine attacks. As a result, we expanded our data collection to include authentication and logon logs as well.

Secondly, we currently do not gather network logs through tools such as Wireshark [156] or Zeek [98]. The system logs collected from each host al-

ready encompass network connection details, thereby offering comprehensive visibility into the network activities of each host. Nevertheless, incorporating network logs could enhance detection efficiency, contingent upon the design of the detection system, and some APT detection systems exclusively analyze network logs, like [97]. Thirdly, both the MITRE ATT&CK knowledge base [48] and MITRE emulation plans [46] are continually evolving to include new attack techniques. For example, the emulation plans for Turla [157], OceanLotus [158], and BlindEagle [159] were recently introduced, while others have been created for several years. We aim to keep pace with updates of MITRE's emulation plans.

## 6.2 Design

In this section, we first describe the emulation infrastructure on which our attack scenarios are conducted. Then we illustrate what and how logs are collected during attack emulation, and where these logs are shipped to. Last, we discuss log parsing.

### 6.2.1 Emulation Infrastructure

The design of our emulation infrastructure is fundamentally derived from MITRE's emulation plans [46]. Currently, none of the emulation scenarios encompass attack steps targeting industrial control systems (ICS), omitting Operational Technology (OT) devices within MITRE's emulation infrastructure. Nevertheless, APT attacks on ICS and OT networks have been notably frequent in recent history [54], [149]–[154]. Our objective is to develop a dataset that encompasses data related to attacks on these networks, thereby fostering research into the detection and investigation of APT attacks on ICS. Consequently, we extend our prior emulation infrastructure to incorporate a representative ICS that includes both an OT network and a field device network.

#### 6.2.1.1 Original MITRE Emulation Plans

MITRE has released eleven full emulation plans [46], which aim to reproduce the attack behaviors of various APT groups observed in real-world scenarios.

These plans are designed to assess and enhance security products from a threat intelligence standpoint. Each plan is based on threat reports that detail the attack tactics and techniques associated with a particular APT group. The emulation plans accurately mirror the actual attack campaigns of these APT groups by adhering to both the overarching stages of the attacks and the specific procedures involved, including the specific actions, commands, tools, and the sequence in which these actions are executed by the attackers.

Among the eleven emulation plans, only two present simplified attack scenarios: the OceanLotus [158], and BlindEagle [159] emulation plans, which target only one or two unmanaged workstations. The remaining plans focus on intricate and realistic cross-machine APT attacks within networks comprised of machines integrated into a Windows Domain through Microsoft Active Directory, reflecting a typical enterprise environment. The current version of AVIATOR includes data gathered from the emulation of four original MITRE plans: APT29 [86], Sandworm [87], WizardSpider [88], and Oilrig [89]. While APT29 and Sandworm are recognized as the most "persistence" APT groups [18], the WizardSpider and Oilrig emulation plans are characterized by their highly intensive Active Directory-based cross-machine attacks.

#### 6.2.1.2 Extended MITRE Emulation Plans

Due to the absence of attack steps on ICS in original MITRE emulation plans, we resort to threat reports linked in MITRE's knowledge base [48] and the MITRE ATT&CK Matrix for ICS [51] to find relevant attack steps in ICS environments, and hence extend MITRE emulation plans. Currently, our dataset AVIATOR includes data collected from emulating two extended MITRE emulation plans, i.e., the extended Sandworm and Oilrig emulation plans. The Sandworm and Oilrig APT groups are best known for their attacks against industrial-sector organizations. Whereas Sandworm is responsible for the attacks against Ukrainian power grid in 2015, 2016 and 2022 [54], [149]–[151], Oilrig has attacked energy and chemical sectors in Middle Eastern in 2017 [152]–[154]. The emulation infrastructures for the two extended MITRE emulation plans are depicted in Figure 6.1 and Figure 6.2, respectively, in which we also show the comparison between the original emulation plans (in dashed grey bounding boxes) and our extended emulation plans.

In our extended emulation plans, we incorporate a gateway or jump server, an engineering workstation, and an OT domain controller through which all

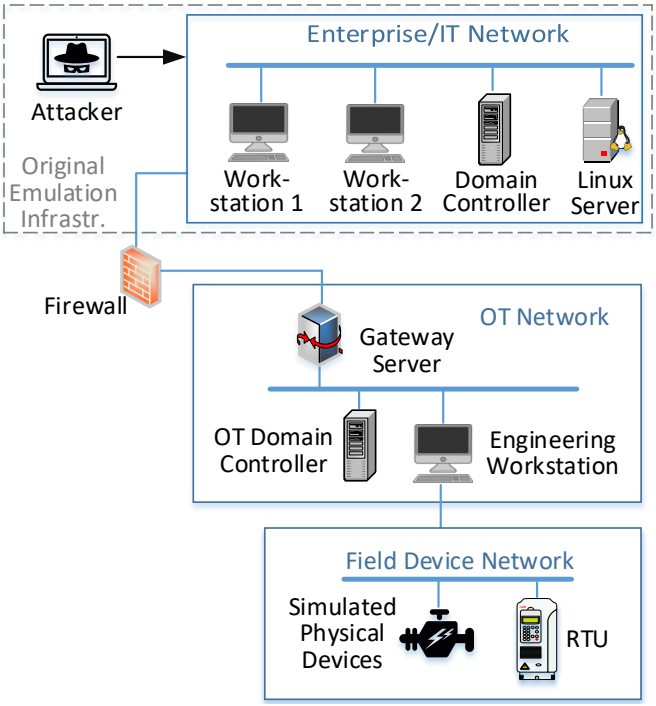



**Figure 6.1:** Extended Sandworm emulation infrastructure.

devices in the OT network are managed. Additionally, we include several standard field controllers, such as the Siemens SIPROTEC [49] and Hitachi Energy RTU500 [50], along with simulated physical devices. Typically, the ICS/OT network of an organization in the industrial sector is isolated from the enterprise or IT network, and managed by dedicated OT domain controllers [41], [160], [161]. A gateway server often facilitates communication between these two networks. The primary targets for attackers aiming at an ICS are generally the engineering workstations, which are used for programming and configuring field controllers, as well as the field controllers themselves, including Programmable Logic Controllers (PLC), Remote Terminal Units (RTU), and Intelligent Electronic Devices (IED), which are responsible for controlling physical processes.

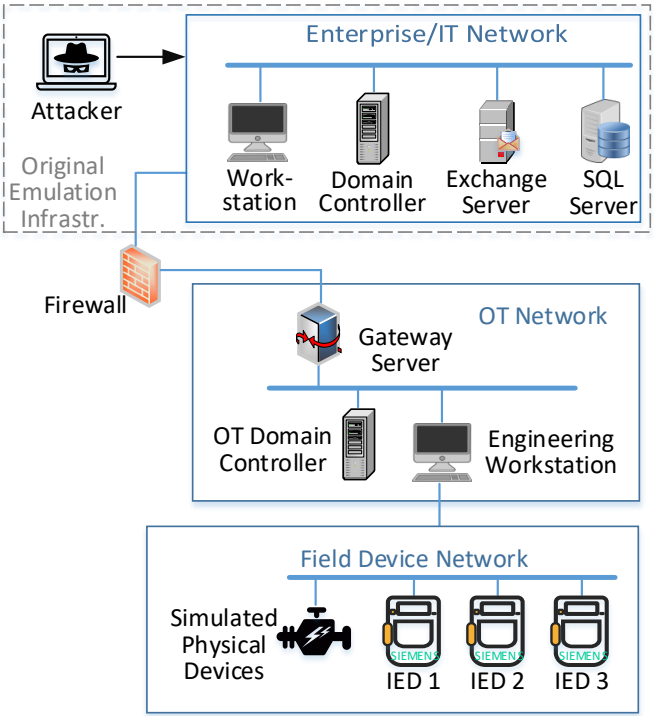



**Figure 6.2:** Extended Oilrig emulation infrastructure [P7].

### 6.2.2 Logging Infrastructure

The MITRE emulation plans are designed purely as an offensive security [162] guidance or a red team [148] tool. Therefore, they do not provide recommendations or instructions pertaining to auditing and logging practices. Instead, the responsibility for establishing and configuring a logging infrastructure is delegated to developers of detection systems. The creation of an effective logging infrastructure can be a labor-intensive endeavor, necessitating a comprehensive understanding of system operations and the interactions of the underlying operating system, along with a certain degree of familiarity with various logging tools. This understanding is crucial for determining what data to log, where to log them, and how to log them. We argue that utilizing predominantly industry-standard logging tools can enhance the practicality

and acceptance of a security system for APT detection and investigation. Therefore, our logging infrastructure primarily incorporates logging tools that are widely utilized across the industry.

In reference to our emulation infrastructure illustrated in Figure 6.2, we present our logging infrastructure in Figure 6.3, detailing the logs collected from various hosts and where they are shipped to. First, we gather authentication logs from the domain controllers, as authentication attempts using domain credentials are exclusively recorded in domain controllers. Conversely, each logon event, which occurs following a successful authentication against a domain controller, is recorded on the accessed host. Thus, we also collect logon logs from individual hosts. Authentication logs and logon logs can be correlated through a logon GUID (Globally Unique Identifier). Additionally, we collect system logs and application logs from the hosts on an individual basis.

On Linux systems, we utilize Auditd [85] for the collection of system logs. In contrast, on Windows systems, our main tools for gathering system logs include Sysmon [84] and the Windows Security Log [155]. Unlike the Windows Security Log and Windows ETW [83], Sysmon is capable of generating and recording a process GUID for each process, which helps to minimize false dependencies during the post-processing of logs. However, it is important to note that Sysmon does not capture significant system activities such as file or Registry reads; therefore, we rely on the Windows Security Log as well. Additionally, we gather ALPC (Asynchronous Local Inter-Process Communication) logs through the "NT Kernel Logger" [163] ETW session. ALPC serves as a method for inter-process communication on Windows, and its operations are not recorded by either Sysmon or the Windows Security Log. Apart from leveraging these standard logging tools, one may write customized code to hook on some standard Windows API (Application Programming Interface), e.g., `CreateProcessA` [164], or register callback functions provided by Microsoft, e.g., `PsSetCreateProcessNotifyRoutine` [165], to obtain host logs.

After deploying those tools for collecting system logs, we found that execution traces of native shell built-in commands, frequently exploited by attackers, are absent from the system logs. Unlike executable-based shell commands, these built-in commands do not generate any log entries. For instance, the execution of Windows PowerShell `cmdlets` [166], which are implemented in Dynamic-Link Library (DLL) files, is not recorded in Sysmon

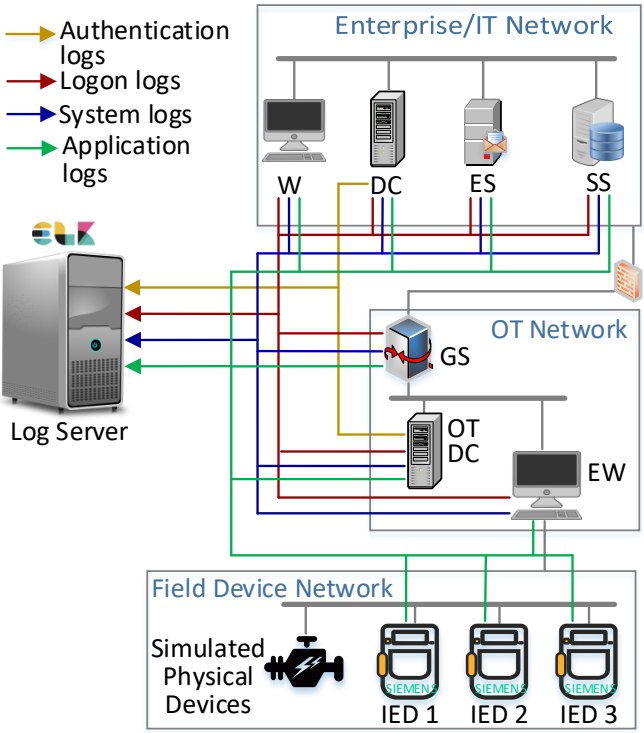



**Figure 6.3:** Logging infrastructure [P7].

or Windows Security logs. Similarly, built-in Linux shell commands, such as `echo`, do not appear in Auditd logs. This lack of logging prompted us to enhance system logs with application-specific logs, such as PowerShell logs on Windows. By incorporating these application logs, we are able to capture essential connections and attack steps, thereby improving the precision and effectiveness of APT detection and investigation.

In the context of field controllers, achieving fine-grained logging is often impractical due to the real-time constraints of these systems. Consequently, we primarily collect logs generated by the devices themselves, treating them as application logs. From a log granularity perspective, we also categorize authentication and logon logs as application logs; however, due to their distinct capability in accurately identifying cross-machine attacks, we designate them

their own specific categories. To maintain log integrity, recent versions of Sysmon operate as protected processes, which helps prevent tampering or disabling by malicious actors. Additionally, Windows has introduced *Protected Event Logging* [167], which employs cryptographic measures to safeguard logs from unauthorized access. On the Linux platform, making Auditd configuration immutable [168] provides protection against unauthorized changes to rules and disabling without a system reboot.

### 6.2.3 Data Shipping

Logs gathered from each device are transmitted to a centralized log server where Elasticsearch [169], Logstash [170] and Kibana [171] services are running. Collectively referred to as the ELK stack, these services are intended for the processing, searching, analyzing, and visualizing of data from various sources and formats. On Linux systems, we have deployed Auditbeat [172] and Filebeat [173] to transmit Auditd logs either at the moment of emission or after they are written to a file, respectively. For Windows systems, Winlogbeat [174] is utilized to send Sysmon logs and Windows Security logs. All application logs from each host are forwarded using Logstash, as these logs tend to be less standardized and cannot be processed on the fly by Auditbeat, Filebeat, or Winlogbeat. We have created a configuration file for each type of application log to ensure that Logstash can accurately convert and forward the logs to the Elasticsearch database.

### 6.2.4 Data Parsing

Elasticsearch offers a scalable and near real-time search capability for the investigation of log data. The process of Elasticsearch on the log server is accessed through Event Query Language (EQL) [175], which is a query language tailored specifically for security applications. To effectively interpret the logs that have been collected and transmitted, it is essential to first parse them individually and then analyze them in aggregate. We have created a distinct parser in EQL for each log type corresponding to the various logging tools. This is necessary because each logging tool generates log events with a unique set of attributes, requiring a specialized parser for each. Additionally, the parsed logs can be input into a program we developed for the on-demand

generation of provenance graphs and their visualization, utilizing Python NetworkX [176] and PyVis [177], respectively.

## 6.3 Dataset Quality Criteria

To facilitate the comparison of existing datasets and to assist in the creation of new ones, we propose three primary criteria for evaluating the quality of a dataset intended for APT detection and investigation: 1) the complexity and authenticity of attack scenarios, 2) the operability and usability of the dataset, and 3) the reproducibility and extensibility of the dataset.

### 6.3.1 Complexity & Authenticity of Attack Scenarios

While many datasets are presented by their creators as representative of APT attacks in real world, we observe that certain datasets are derived from highly shortened and excessively simplified attack scenarios, which do not accurately portray realistic APT attacks. For an attack scenario to be considered appropriate for research on APT detection and investigation, it is essential that it fulfills certain minimum criteria regarding the tactics and techniques employed, as well as the emulation infrastructure utilized.

**APT Life Cycle.** While certain attack scenarios encompass a comprehensive attack chain, others are limited to one or a few stages of the attack. Among all the stages, we regard the cross-machine attack stage, the stealthy/persistence attack stage, and the evasive attack stage as the most critical components of a realistic APT attack. Consequently, we designate a dataset as representing a complete APT life cycle when it contains at least one cross-machine attack stage and at least one of either a stealthy attack stage or an evasive attack stage.

**Stealthy Attack Techniques.** A defining characteristic of APT actors is their adoption of the low-and-slow strategy. Whereas these attackers utilize Living-Off-the-Land Binaries (LOLBins) [63], [64] to stay low, they employ persistence techniques to stay slow, thereby enhancing their stealthiness. Most APT groups adopt persistence attack techniques to ensure continued access to compromised systems, which allows them to conduct extended attack campaigns against their targets [18].

**Evasive Attack Techniques.** Another notable characteristic of APT actors is their ability to evade detection. Unlike conventional hit-and-run attacks, APT attacks can remain undetected for extended durations, facilitated by various evasive techniques such as disabling security software, disguising harmful processes, and eliminating traces of their activities.

**Cross-host Attack.** In practical scenarios, APT actors usually gain initial access through a more vulnerable machine and must move laterally to other systems that contain critical data of the targeted organization. Lateral movement between machines is a common feature in authentic APT attacks. Typically, to access the ultimate target machine, attackers must perform lateral movement multiple times, owing to the network segmentation that is often employed within organizations' networks.

**Host Number.** A standard target network for APT attacks typically comprises a minimum of a dozen hosts, rather than merely one or two. Furthermore, these hosts are interconnected, either directly or indirectly.

**Host OS.** Given their significant market share as desktop operating systems and server operating systems, it is essential to consider both Windows and Linux distributions in the context of emulated APT attacks.

**Host Type.** In addition to the workstations utilized by employees, an organization typically maintains a range of application servers, such as email servers, web servers, and data servers. These servers are frequently considered more valuable targets for APT actors compared to workstations. Among these, domain controllers stand out as the most critical and desirable systems, as their compromise can lead to devastating consequences for the organization.

**Domain Joined.** In modern enterprises' IT infrastructures, devices are frequently integrated into a domain to facilitate centralized identity and access management (IAM). IAM systems are prevalent within corporate networks, with Microsoft Active Directory being the most widely utilized solution. The existence of an IAM system, such as an Active Directory domain controller, significantly influences the tactics employed by APT actors, as these systems are primary targets for such adversaries in practice.

**ICS Included.** Corporate networks have historically been the focus of APT actors; however, there has been a notable rise in the targeting of ICS networks as well [54], [149]–[154]. Incorporating an ICS into emulation plans

significantly enhances the dataset's applicability for research related to APT detection and investigation.

**Data Size.** The size of data is influenced not only by the duration of data collection but also by the level of log granularity. In the context of APT detection and investigation, application logs have limited utility, as these logs, characterized by their coarse granularity, do not adequately establish causal relationships between the malicious activities of APT actors. Furthermore, they fail to provide the necessary context for effective attack detection and investigation. Conversely, fine-grained system logs are crucial for the development of causality-based provenance graphs.

**Duration.** We believe that a research dataset focused on APT detection and investigation ought to encompass both benign and malicious activities over a duration of no less than one week.

### 6.3.2 Dataset Operability & Usability

To effectively enhance the utilization of a dataset, it is essential to be easily operable and usable. This entails the inclusion of the following components: 1) a diverse range of log types, 2) a ground truth for the attack steps executed during the emulation process, 3) mechanisms for transferring logs to a centrally managed log database, and 4) tools for searching and parsing logs to facilitate APT detection and investigation.

**Log Types.** In our research on APT detection and investigation, we have observed that logs from various types and sources make complementary contributions in uncovering the extent of APT actors' traces within computer systems and networks. While system logs are widely regarded as the most significant resource for APT detection and investigation [22]–[26], [29], [33], [35], [36], [80], [81], they are insufficient on their own for detecting attacks that span multiple machines. We find that authentication and logon logs are especially beneficial for precise and effective identification of cross-machine attacks.

**Ground Truth Provided.** Evaluating an APT detection system on a dataset becomes significantly more challenging and time-consuming when the ground truth is not available. However, it is noteworthy that two-thirds of the datasets analyzed in Table 6.1 do include a ground truth.

**Log Shipping Tools Provided.** The provision of shipping tools or shippers, such as Logstash [170] and Filebeat [173], along with their respective configuration files for the collected logs, significantly minimizes the time required for preparing log analysis.

**Log Parsing Tools Provided.** Creating parsers for different types of logs is a demanding task. With the availability of parsing tools, researchers are able to concentrate on their primary responsibilities, such as log analysis and the development of detection schemes, thereby greatly accelerating the process of APT detection and investigation.

### 6.3.3 Dataset Reproducibility & Extensibility

A dataset that is both reproducible and extensible is deemed to offer greater value to the research community. This is due to the fact that researchers may have varying scopes or focuses in the detection and investigation of APT attacks, or their detection algorithms may necessitate log sources that are not included in the existing dataset.

**Emulation Infrastructure Provided.** The provision of emulation infrastructure, e.g., as virtual machine images or snapshots, represents the most efficient method for other researchers to reimplement the emulation plans, potentially with enhancements in the complexity and authenticity of attack scenarios, as well as improvements in auditing and logging. Nevertheless, this approach is not feasible for Windows hosts due to copyright and licensing constraints.

**Logging Infrastructure Provided.** Establishing a sound logging infrastructure may require as much time as developing the emulation infrastructure. By supplying logging configuration files, a significant amount of time can be conserved for other researchers.

**Actively Maintained.** Dataset creators may also choose to actively update their datasets by taking into account requests from fellow researchers to include additional attack techniques or log sources in subsequent iterations of the datasets.

## 6.4 AVIATOR vs. Prior Datasets

We compare our dataset AVIATOR with existing datasets in Table 6.1[1]. We find that creators of more than half of these datasets implemented only a partial APT life cycle. Less than one-third of the prior datasets include a stealthy attack step or evasive attack step, while a cross-machine attack stage is observed in more than two-thirds of them. Half of these datasets were created from an emulation infrastructure consisting of only one or two hosts. Windows is the mainly targeted OS, present in most datasets. Each dataset includes logs from at least one workstation-level host, while logs from application server-level hosts are present in half of the datasets. Apart from our dataset AVIATOR, only the DARPA-OpTC [71] dataset's emulation infrastructure contains a domain controller, through which the hosts are joined into a domain and managed centrally, as in a realistic enterprise setting. None of the existing datasets' emulation infrastructure incorporates an ICS network.

[1] Acti. = Actively, Andr. = Android, Appli. = Application, AS = Application Server, Auth. = Authentication, cpr. = compressed, DC = Domain Controller, Dura. = Duration, EOS = Embedded OS, Emu. = Emulation, Evasi. = Evasive, exten. = extensibility, GB = Gigabyte, Grou. = Ground, IC = Industrial Controller, incl. = included, infr. = infrastructure, Log. = Logging, main. = maintained, MB = Megabyte, Mobi. = Mobile, Netw. = Network, num. = number, pars. = parsers, part. = partial, prov. = provided, reprod. = reproducibility, Pub. = Publication, ship. = shippers, sin. = single stage, Steal. = Stealthy, TB = Terabyte, tech. = technique, Win. = Windows, Work. = Workstation.

**Table 6.1:** Comparison of AVIATOR and prior datasets for APT detection and investigation [P7].

| Dataset | Pub. year | Attack scenario complexity & authenticity | | | | | | | | | | | Dataset operability & usability | | | | Dataset reprod. & exten. | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | APT life cycle | Steal. attack tech. | Evasi. attack tech. | Cross host attack | Host num. | Host OS | Host type | Domain joined | ICS incl. | Data size | Dura. | Log types | Grou. truth prov. | Log ship. prov. | Log pars. prov. | Emu. infr. prov. | Log. infr. prov. | Acti. main. |
| AVIATOR | 2024 | full | ✓ | ✓ | ✓ | 12 | Win. Linux EOS | Work. AS DC IC | ✓ | ✓ | 1.9 TB | 2 weeks | System Appli. Auth. Logon | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ |
| Linux -APT | 2024 [125] | sin. | ✓ | ✓ | ✓ | 2 | Linux | Work. | ✗ | ✗ | 205 MB | 6 weeks | Appli. | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| ATLASv2 | 2023 [124] | part. | ✗ | ✗ | ✓ | 2 | Win. | Work. | ✗ | ✗ | 154 GB | 5 days | System Appli. Netw. | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
| Public-Arena | 2023 [34] | full | ✗ | ✓ | ✓ | 2 | Win. | Work. AS | ✗ | ✗ | 7 GB | 6 days | System Appli. | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
| Unraveled | 2023 [123] | full | ✗ | ✓ | ✓ | 27 | Win. Linux | Work. AS | ✗ | ✗ | 1 TB | 6 weeks | System Appli. Logon Netw. | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| ATLAS | 2021 [32] | part. | ✗ | ✗ | ✓ | 2 | Win. | Work. | ✗ | ✗ | 6.5 GB | 1 day | System Appli. Netw. | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
| DAPT-2020 | 2020 [122] | part. | ✗ | ✗ | ✓ | 4 | Linux | Work. AS | ✗ | ✗ | 10 GB | 1 week | System Appli. Logon Netw. | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| DARPA-OpTC | 2020 [71] | full | ✓ | ✗ | ✓ | 28 (/500) | Win. | Work. DC | ✓ | ✗ | 1TB (cpr.) | 1 week | System Netw. | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
| DARPA-E5 | 2019 [70] | part. | ✓ | ✗ | ✗ | 18 | Win. Linux BSD Andr. | Work. AS Mobi. | ✗ | ✗ | 340 GB (cpr.) | 9 days | System | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ |
| DARPA-E3 | 2018 [69] | part. | ✗ | ✗ | ✗ | 7 | Win. Linux BSD Andr. | Work. AS Mobi. | ✗ | ✗ | 116 GB (cpr.) | 10 days | System | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ |
| StreamSpot | 2016 [116] | part. | ✗ | ✗ | ✗ | 1 | Linux | Work. | ✗ | ✗ | 2 GB | 1 day | System | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |

Only the StreamSpot [116] and ATLAS [32] datasets consist of logs of just one day's operation, all other datasets contain logs of operation spanning close to or more than a week. All datasets but the Linux-APT [125] dataset include system logs. This is the reason that the Linux-APT dataset has a size of roughly 200 Megabytes, while all other datasets hold Gigabytes or even Terabytes of data. About a third of prior datasets do not come with a ground truth explaining what exactly the commands or actions undertaken during the attack scenario emulation are, and in which order. The DARPA-E3 [69] and DARPA-E5 [70] are the only two existing datasets that provide log shipping tools to ingest the data to a centrally managed log server. Among all datasets in Table 6.1, our dataset AVIATOR is the only dataset in which log parsing tools are provided. Further, AVIATOR is also the only dataset in which the logging configuration files are provided. We intended to publish our emulation infrastructure as well. Unfortunately, due to Microsoft's policies on redistribution of Windows, this is not possible. However, we are committed to maintaining AVIATOR as a living dataset.

The earliest dataset in Table 6.1, i.e., StreamSpot [116], was created to evaluate the anomaly-based APT detection system published in the same article [116], and chosen for evaluation of state-of-the-art APT detection systems like [28], [35], [36]. Yet, this dataset consists of only a simplified, partial APT attack chain, and does not cover any of persistence, defense evasion and lateral movement attack stages. The attack scenario in this dataset is confined in a single Linux machine.

The second earliest dataset in Table 6.1, i.e., DARPA-E3 [69], is one of the most popular datasets for evaluation of provenance-based APT detection and investigation systems [28], [29], [33]–[35], [81], [115]. This dataset was created in the DARPA Transparent Computing (TC) program [68], and released by DARPA to stimulate further research on APT detection and investigation. However, like the StreamSpot [116] dataset, this dataset consists of only partial APT attack chains, and does not include any of persistence, defense evasion and lateral movement attack stages. Although the DARPA-E3 dataset's emulation infrastructure includes seven hosts of various OS types and host types, these host were not connected in a network, but rather targeted individually. The dataset holds more than 100 Gigabytes of compressed data, which are more than ten times bigger when decompressed.

The DARPA-E5 [70] dataset is another dataset generated from the DARPA TC program, but during the Engagement 5, instead of the Engagement 3. This

dataset improves on the DARPA-E3 dataset, by increasing the attack scenario complexity, e.g., adding two persistence attack steps, and the number of hosts in the emulation infrastructure. Yet, neither defense evasion attack stages nor lateral movement stages are included in the attack scenarios, hindering the practicality of this dataset.

To enhance the DARPA TC program, the Operationally Transparent Cyber (OpTC) study was undertaken, during which the team expanded the emulation infrastructure to encompass one thousand machines. Notably, these machines are integrated within a Windows Domain. Additionally, the team executed persistence and cross-machine attacks throughout the attack emulation, thereby simulating a complete APT life cycle. Due to limitations in storage capacity, only half of these machines were actively monitored, resulting in the generation of logs for the dataset known as DARPA-OpTC [71]. Our analysis of this dataset indicates that, out of the 500 monitored hosts, merely 28 were engaged in the emulated attack scenarios. In comparison to the DARPA-E3 and DARPA-E5 datasets, the operating system scope of the DARPA-OpTC dataset is exclusively focused on Windows. Furthermore, this dataset also comprises network logs, specifically Zeek logs.

The DAPT-2020 dataset [122] was developed to facilitate research on APT detection, primarily from a network-oriented perspective. This dataset was generated within an environment exclusively comprising Linux machines. While the dataset includes system logs, the predominant portion consists of network logs. The collection of system logs appears to have been conducted at a basic level, as indicated by the overall data volume. It seems that the dataset creators utilized the default settings for system log collection, which do not account for critical and log-intensive system activities, such as process creation and file access. Consequently, this limitation restricts the dataset's utility for research focused on provenance-based APT detection and investigation.

Both the ATLAS [32] dataset and the Public-Arena [34] dataset were designed specifically for the training and evaluation of a provenance-based detection system. The emulation infrastructure for each dataset comprises solely two Windows machines. While both datasets feature cross-machine attack stages, the Public-Arena dataset includes a defense evasion attack stage. It is important to highlight that the Public-Arena dataset contains only a persistence setup, lacking any persistence execution. Such a step can be interpreted as

an unsuccessful attempt to carry out a persistence attack technique, as we discussed in Section 4.2.

The Unraveled [123] dataset presents a sophisticated and seemingly authentic corporate network setup. However, it remains ambiguous which specific hosts are involved in the attack scenarios, as the dataset and the accompanying publication do not provide the ground truth. Instead, a brief overview of each attack is included, lacking any reference to particular hosts. Our thorough review of the logs within the published dataset reveals that the hosts are not even joined in a domain. Similar to the DARPA-OpTC dataset, the large number of hosts in this dataset is primarily due to the replication of a limited number of hosts. In contrast, each host in our dataset AVIATOR serves a unique function rather than being a mere duplicate. Furthermore, akin to the DAPT-2020 dataset, the system logs in the Unraveled dataset constitute only a minor fraction of the overall dataset, with the majority comprising network logs.

In a similar emulation infrastructure setting for the ATLAS dataset, the ATLASv2 [124] dataset was developed by different creators with the objective of enhancing the original ATLAS dataset. This enhancement is achieved by incorporating additional background noise, specifically several days of benign system activities performed by legitimate users prior to the attack scenarios. Furthermore, the ATLASv2 dataset comprises system logs gathered from Sysmon, alongside Windows Security logs and application logs produced by VMware Carbon Black Cloud. Nevertheless, it is important to note that the creators did not augment the complexity or authenticity of the attack scenarios, which remain identical to those found in the ATLAS dataset.

In the latest dataset pertaining to research on APT detection and investigation Linux-APT [125], the developers gathered and released solely the security alerts generated by Wazuh [178], amounting to a total of 200 Megabytes of data collected over a span of six weeks. We categorize the logs or alerts produced by a security application, rather than a dedicated auditing tool, as application logs. We believe that application logs *alone* hold limited value for research on APT detection and investigation due to their coarse granularity. Furthermore, the dataset creators did not provide the ground truth, nor did they make available the configuration file for the Wazuh agent program, which complicates the interpretation and utilization of the dataset.

## 6.5 Summary

We provide in the present chapter a novel dataset called AVIATOR for stimulating further research on APT detection and investigation. AVIATOR is based on both original and expanded MITRE emulation plans, which target realistic enterprise and ICS networks. These emulation plans incorporate more advanced, cross-machine attack scenarios compared to those found in previous datasets. AVIATOR surpasses existing datasets due to its high complexity and authenticity of attack scenarios, as well as its operability, usability, reproducibility, and extensibility. In future work, we intend to automate the execution of these emulation plans. However, automating these plans presents challenges and is time-intensive, as certain hacking tools referenced in the MITRE emulation plans are not inherently designed for automation. For example, when a callback is received from a command and control (C2) server, a random session number may be generated. To effectively automate the C2 commands, this session number must be predictable, requiring tool modification and recompilation, which significantly extends the time required for data generation. Currently, we have achieved automation for the APT29 emulation plan. Additionally, we are developing solutions to automate the creation of our emulation infrastructure using tools such as Terraform and Ansible, with the goal of publishing our emulation infrastructure configuration files in future iterations of AVIATOR. Furthermore, we plan to incorporate network logs, including complete PCAP and Zeek logs, in future versions of AVIATOR.

# 7 Cyber Persistence Detection and Investigation

In Section 2.1, we empathize that the most symbolic behavior of Advanced Persistent Threat (APT) actors is staying low and slow to avoid detection. Whereas employing Living-Off-the-Land Binaries (LOLBins) is the most common way to stay low, persistence techniques are routinely applied to perform slow attack process, in which an entire attack chain is deliberately broken down into multiple seemingly unrelated phases. In this chapter, we introduce our first APT detection and investigation system CPD, which is, to the best of our knowledge, the first specialized system for persistence detection and investigation. CPD provides a comprehensive solution for detecting true persistence attacks and eliminating most false positives, therefore empowering connection of the "big dots" and realizing multi-phase APT detection and investigation.

## 7.1 Challenges & Objectives

In tackling the intricacies associated with APT attacks, Provenance-based Intrusion Detection Systems (PIDS) [22]–[26], [28], [29], [33], [35], [36], [80], [81] have become a vital tool by parsing system audit logs into provenance graphs, and providing causal dependencies between system entities,

This chapter is based on a previous paper of ours. Note that, to differentiate our own works from others, we cite our own works with a leading letter "P". Inside the chapter, these kind of citations are used in particular for tables, figures and algorithms taken or adopted from our previous paper:

[P4] Q. Liu, M. Shoaib, M. U. Rehman, K. Bao, V. Hagenmeyer, and W. U. Hassan, *Accurate and scalable detection and investigation of cyber persistence threats*, 2024. arXiv: 2407.18832 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2407.18832, under review

such as processes, files, Registry entries, and network sockets. In contrast, rule-based intrusion detection systems, for instance, Elastic [146], Google Chronicle [75], and Sigma [74], which match system logs against a collection of predetermined signatures, are standard ways in industry to identify persistence threats.

As characterized by MITRE [48], persistence techniques commonly entail the exploitation of "sensitive" system functionalities, such as Registry Run keys / Startup Folder [132]. Existing attack detection systems issue alerts for persistence attacks, each time these "sensitive" system functionalities are accessed, regardless of whether such access is being exploited by an attacker or legitimately utilized by an authorized user. This approach does not evaluate the implications of using persistence-related system functionalities, which may only become evident at a later stage. As a result, on the one hand, this often leads to a significant number of false alarms; for instance, a genuine user who adds entries to the Registry Run keys or startup folders to enable programs to launch upon log-on would trigger an alert, despite the action being actually harmless. On the other hand, if an APT actor drops a Command & Control (C2) agent program into a startup folder, or configures a Registry Run key to activate the C2 agent program that automatically reconnects back to the attacker after a reboot, current systems may detect the creation of a new file in the startup folder, or the creation of the Registry Run key, but fail to associate it with subsequent C2 activities due to the time lag in their occurrence and the absence of a thorough contextual analysis required for precise persistence detection.

As discussed in Section 4.2, existing PIDS fail to accurately detect persistence due to a lack of understanding of persistence attacks' semantics. Existing rule-based persistence detection systems, such as well-known solutions like Elastic [146] and Google Chronicle [75], suffer from a considerable incidence of false positives.These systems generally assess persistence techniques in isolation, overlooking the wider context of an attack. Consequently, they frequently trigger alerts for activities that may seem suspicious but are, in fact, harmless, resulting in a plethora of false alerts. To alleviate these false positives, these systems may excessively relax their detection criteria. For example, our analysis in Section 7.4 indicated that actions from applications located in standard directories are automatically considered as benign without adequate examination. This practice has unintentionally led to a notable rise in false negatives. The narrow detection strategy has practical implications in Security Operations Centers (SOC), where security analysts are

inundated with alerts and must establish a threshold to filter out the majority. This highlights the urgent necessity for a system capable of automatically decreasing the volume of alerts.

In response to the challenges associated with existing persistence threat detection methods, we present Cyber Persistence Detector (CPD), a novel system designed to facilitate rapid and precise identification of persistence threats within enterprise networks. Our approach aims to avoid optimistic assumptions about persistence behavior, generate few false positives and false negatives, triage persistence-related threat alerts, and generate accurate graphs for swift attack investigation.

We find that effective persistence attacks always consist of two phases: the persistence setup (e.g., dropping a C2 agent program into a startup folder, or creating a Registry Run key that specifies an arbitrary location for the C2 agent program), and the persistence execution (e.g., a remote connection initiated by the C2 agent program dropped in the startup folder or associated with the newly created Registry Run key). *Persistence setup serves solely as preparation, whereas persistence execution exhibits attackers' true motives.* Leveraging this key insight, CPD introduces a novel concept called *pseudo-dependency edges* (pseudo-edges) define them as follows:

**Definition 1** *pseudo-edges are edges that are specifically created to connect a persistence setup's system activities with the corresponding persistence execution's system activities. Unlike normal edges, these edges cannot be created by parsing system logs alone.*

Nonetheless, relying exclusively on pseudo-edges to improve the accuracy of persistence detection poses several challenges. A significant issue is that even the most thorough logging systems may fail to capture every connection, resulting in gaps within the provenance graph and, as a result, leading to false negatives. Our research indicates that the integration of Windows ALPC (Asynchronous Local Inter-Process Communication) logs with system audit logs can help address these gaps. However, this integration significantly escalates the storage requirements for logs. Rather than relying on ALPC logs, we propose a novel technique in CPD aimed at refining the provenance graph while simultaneously reducing log volume. Furthermore, the behavior of certain benign applications can lead to an overproduction of pseudo-edges, complicating the detection process. To address this challenge, we aim to

incorporate a false positive reduction algorithm in CPD, which will ensure that only genuinely malicious activities are flagged.

## 7.2 CPD Design

CPD employs a four-step approach for effective persistence detection while aiming to minimize false positives, as illustrated in Figure 7.1. Initially, CPD processes system event logs to create a *persistence setup table*, recording activities that match persistence setup detection rules, such as Registry Run key creations or additions. Then, it pinpoints processes with remote connection(s) in the event stream, and performs individual backward tracing to generate sub-graphs for each of these processes. The sub-graphs are subsequently checked against our persistence execution detection rules, and the matches are recorded in a *persistence execution table*. In the third step, CPD aligns the entries from the persistence execution table with the entries from the persistence setup table based on TTP labels, temporal order, and some attributes specific to a TTP. When an alignment is found, CPD creates a *persistence setup atomic graph*, i.e., a minimal sub-graph directly related to persistence setup, and a *persistence execution atomic graph*, i.e., a minimal sub-graph directly related to persistence execution. It then links them with a pseudo-edge. In the evaluation section (Section 7.4), steps two and three are combined into *stage two*, since step two alone does not yield direct alerts/results. The final step introduces *pseudo-edge strength* and a false positive reduction algorithm to make sure that only significant events are connected.

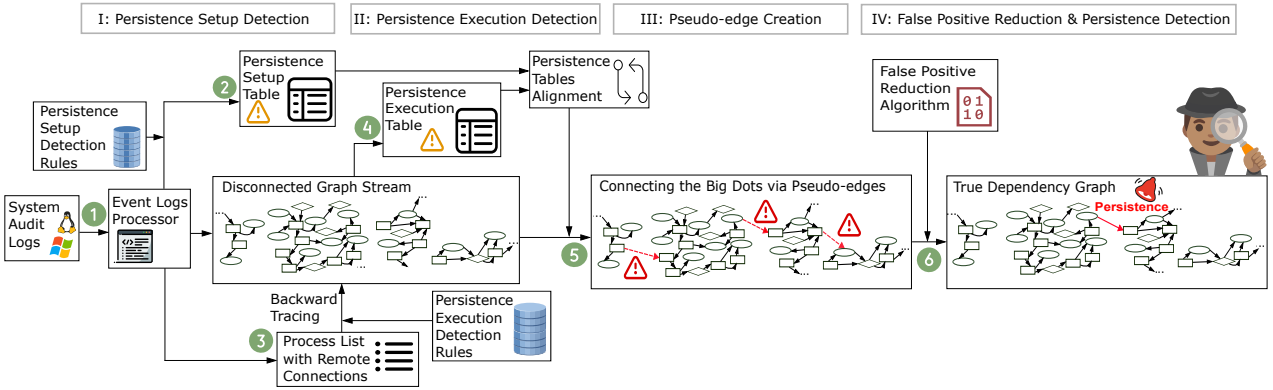



**Figure 7.1:** CPD overview [P4]. CPD implements a four-step approach for detecting persistence threats, starting with the creation of a persistence setup table from audit logs that tracks potential setup actions. It then traces processes with remote connections to form sub-graphs, which are evaluated against execution rules and aligned with setup actions to form atomic graphs linked by a pseudo-edge. The process is refined through the introduction of pseudo-edge strength and a false positive reduction algorithm.

### 7.2.1 Persistence Threat Detection

Our rules for detecting persistence setups are formulated through an analysis of persistence (sub-)techniques outlined in the MITRE ATT&CK Matrix, hundreds of threat reports related to persistence, and red team tools that offer insights into low-level code associated with persistence attacks, such as Atomic Red Team [179]. We have further refined our detection rules for persistence setups by examining and integrating open-source detection rules from well-known repositories like Sigma [74] and Elastic [146]. Nonetheless, as elaborated in Section 7.4.1, our rules differ from those found in these repositories. The process of detecting persistence setups primarily involves a straightforward string matching technique applied to data within a single system log event. However, certain rules contain additional conditions that require information from multiple log events. For this purpose, we employ "sequenced" query in EQL, a query language specifically tailored for threat hunting [175]. Key strings of interest include file paths, Registry locations, process names, and command lines, which are particularly indicative of the persistence (sub-)techniques. For example, to implement the persistence sub-technique T1547.001 (Registry Run keys / Startup Folder) [132], one of a few known Registry locations *must* be modified.

The mere act of aligning system events with detection rules results in a substantial number of alerts in practice. To mitigate the occurrence of false alarms, CPD initially identifies every process that initiates or accepts remote connections within the event stream. Subsequently, it conducts a backward tracing of these processes on an individual basis. During this backward tracing, CPD examines whether its provenance graph includes system activities that correspond to our persistence execution detection rules. Similar to the rules for detecting persistence setups, our detection rules for persistence executions encompass various indicative strings, which include process names, file paths, Registry locations and more. These strings are also representative of the persistence (sub-)techniques. For instance, to "activate" the persistence sub-technique T1547.001 (Registry Run keys / Startup Folder) [132], it is essential that one of several known Registry locations is accessed specifically by the process `explorer.exe`, and it *must* depend on this process to initiate the malicious process, either directly or indirectly.

Table 7.1 presents the sensitivity associated with these detection rules. *It is important to note that we operate under the assumption that the integrity of the*

*operating system, including its native built-in programs, is assured.* That is to say, an intended, persistence-related system functionality, e.g., Registry Run Keys / Startup Folder, will operate as *designed and expected.* We highlight that the first stage of CPD is designed to detect all potential persistence attacks, albeit at the cost of generating numerous false positives, as it prioritizes sensitivity over specificity during this initial detection phase. The optimization techniques outlined in CPD, as discussed in Section 7.4.1, enhance the specificity of stage one without affecting its sensitivity. Furthermore, we contend that the evasion techniques for SIEM (Security Information and Event Management) rules presented in [180] have a limited effect on our detection rules. As indicated in Table 7.1, our approach primarily does not depend on recorded command lines or the code executed by attackers; instead, we focus on indicative file paths, Registry locations, and system process names, which remain unchanged under the assumption of OS integrity.

For each identified potential malicious process exhibiting remote connections, CPD inspects whether there are corresponding entries in the persistence setup table. This assessment is based on TTP labels, specific attributes related to TTP, and the happens-before relationship (refer to Algorithm 1 Lines 4-9). Upon discovering an alignment, a persistence execution atomic graph is generated (Algorithm 1 Line 10), which encompasses only the information pertinent to persistence execution. Similarly, a persistence setup atomic graph is created, focusing solely on critical attack information. Subsequently, CPD establishes a pseudo-edge to link the persistence setup atomic graph with the persistence execution atomic graph (Algorithm 1 Line 12), culminating in a concise and informative persistence attack graph, illustrated in Figure 7.2.

**Table 7.1:** Detection rule sensitivity for the top 10 persistent techniques [P4]. TPR = True Positive Rate, ✓ = Yes, ✓̸ = Almost.

| (Sub-) techniques | Detection rules with TPR=1? | Explanation |
|---|---|---|
| Registry Run Keys / Startup Folder | ✓ | During persistence setup, new entries ***must*** be added to the the standard Registry Run keys locations or standard Startup folders. During persistence execution, the corresponding new entries ***must*** be read by the system process `explorer.exe`. The malicious process ***must*** be ultimately started by `explorer.exe`. |
| Scheduled Task | ✓ | During persistence setup, one of a few Windows task creation programs / Powershell `cmdlets` / API ***must*** be called, and file modification in the Windows standard Tasks folder ***must*** be undertaken. During persistence execution, the malicious process ***must*** be ultimately started by the system process `svchost.exe` with the flags "`-k netsvcs -p -s Schedule`". |
| Web Shell | ✓̸ | During persistence setup, a file containing executable code like PHP is ***very likely*** dropped to the web root directory like `/var/www/html`. During persistence execution, the malicious process ***must*** be ultimately started by the web server program like `apache2`. |
| DLL Side-Loading | ✓ | During persistence setup, a DLL file ***must*** be dropped to the file system. During persistence execution, that DLL file ***must*** be loaded by a process initiating or accepting remote connection(s). |
| External Remote Services | ✓ | During persistence setup, a common remote access program ***must*** be installed, and its executable file ***must*** be dropped to the file system. During persistence execution, that program ***must*** initiate or accept remote connection(s). |
| Windows Service | ✓ | During persistence setup, a new entry ***must*** be added to the standard Registry location for Windows services. During persistence execution, the corresponding new entry ***must*** be read by the system process `services.exe`. The malicious process ***must*** be ultimately started by `services.exe`. |
| Domain Accounts | ✓ | During persistence setup, a new entry ***must*** be created in the domain controller's standard Active Directory database stored in the file system. During persistence execution, this new account ***must*** be used for logging into target systems. |
| WMI Event Subscription | ✓ | During persistence setup, one of a few Windows WMI event creation programs / Powershell `cmdlets` / API ***must*** be called, and file modification in the Windows standard WMI event repository ***must*** be undertaken. During persistence execution, the malicious process ***must*** be ultimately started by the system process `wmiprvse.exe`. |
| DLL Search Order Hijacking | ✓ | During persistence setup, a DLL file ***must*** be dropped to the file system. During persistence execution, that DLL file ***must*** be loaded by a process initiating or accepting remote connection(s). |
| Local Account | ✓ | During persistence setup, a new entry ***must*** be created in the standard user information database stored in the local file system. During persistence execution, this new account ***must*** be used for logging into target systems. |

**Algorithm 1:** PSEUDO-EDGE CREATION [P4]

```
1  Function CreatePseudoEdge(Events ℰ)
       /* Get a list of persistent setup events                          */
2      L_<ℰα,ℒα,𝒯α> ← GetPersistenceSetup(ℰ)
       /* Get a list of processes with remote conn.                      */
3      L_<𝒫γ> ← GetProcessWithRemoteConnection(ℰ)
4      foreach 𝒫_γ ∈ L_<𝒫> do
           /* Get a list of persistent exec.  events                     */
5          L_<ℰγ,ℒγ,𝒯γ> ← GetPersistenceExecution(𝒫_γ)
6          foreach (ℰ_γ,ℒ_γ,𝒯_γ) ∈ L_<ℰγ,ℒγ,𝒯γ> do
7              foreach (ℰ_α,ℒ_α,𝒯_α) ∈ L_<ℰα,ℒα,𝒯α> do
8                  if ℒ_γ == ℒ_α then
9                      if 𝒯_γ > 𝒯_α then
10                         AG_γ ← GetAtomicGraph(ℰ_γ)
11                         AG_α ← GetAtomicGraph(ℰ_α)
                           /* Create a pseudo-edge                       */
12                         PAG(γ,α) ← AG_γ ∪ AG_α
13                         L_<PE,PAG> ← L_<PE,PAG> ∪ < PE(γ,α), PAG(γ,α) >
14                     end
15                 end
16             end
17         end
18     end
19     return L_<PE,PAG>

20 Function GetPersistenceSetup(ℰ)
21     foreach Rule ∈ L_PersistenceSetupRule do
22         forall Condition ∈ Rule do
23             satisfied ← CheckCondition(Condition, ℰ)
24             if satisfied then
25                 L_<ℰα,ℒα,𝒯α> ← L_<ℰα,ℒα,𝒯α> ∪ (ℰ,ℒ,𝒯)
26             end
27         end
28     end
29     return L_<ℰα,ℒα,𝒯α>

30 Function GetPersistenceExecution(𝒫_γ)
31     L_<ℰκ> ← TraversalBackward(𝒫_γ, ℰ)
32     foreach ℰ_κ ∈ L_<ℰκ> do
33         foreach Rule ∈ L_PersistenceExecutionRule do
34             forall Condition ∈ Rule do
35                 satisfied ← CheckCondition(Condition, ℰ_κ)
36                 if satisfied then
37                     L_<ℰα,ℒα,𝒯α> ← L_<ℰα,ℒα,𝒯α> ∪ (ℰ_κ,ℒ_κ,𝒯_κ)
38                 end
39             end
40         end
41     end
42     return L_<ℰγ,ℒγ,𝒯γ>
```

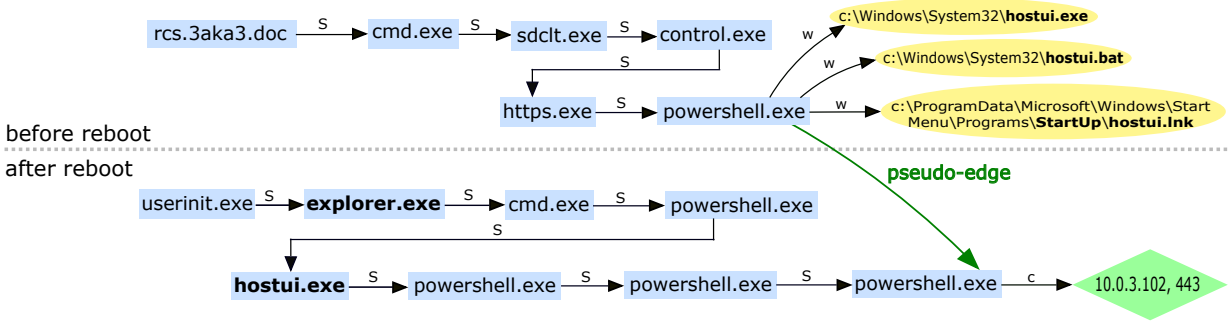



**Figure 7.2:** A persistence attack graph automatically generated by CPD on the AVIATOR-APT29-1 dataset [P4]. It uses rectangles for processes, ovals for files / Registry keys, and diamonds for network sockets. Annotations include S=Start, W=Write, C=Connect. The graph successfully pinpoints T1547.001 (Boot or Logon Autostart Execution: Registry Run keys / Startup Folder). The upper section reveals persistence setup: a malicious Microsoft Word-like program (`.doc`) starts, resulting in a Powershell instance and a shortcut creation in the Windows startup folder. This shortcut leads to another dropped malicious program, `hostui.exe`. The lower section, post-reboot, shows persistence execution: `explorer.exe` auto-executes startup folder shortcuts, triggering malicious Powershell code and connecting to the attacker. Indicative strings are bolded for clarity. CPD forms a pseudo-edge linking the process initiating persistence setup with the one managing the remote connection, i.e., the C2 agent.

### 7.2.2 Expert-guided Edges

Our research revealed certain limitations in the ability to link system entities exclusively through the logs generated by Windows' Process Monitor [181] and System Monitor [84]. In particular, during the execution of T1543.003 (Create or Modify System Process: Windows Service) [120] for establishing persistence, attackers frequently utilize the `sc.exe`[1] program to create a harmful Windows service. This action leads to the creation of a new Registry key located at `HKLM\SYSTEM\CurrentControlSet\Services`, which serves as the foundation for our detection rule. This method, which emphasizes an immutable Registry location, proves to be more reliable than depending on command lines, which can be easily circumvented, as indicated by recent research [180].

The logs indicate that the modifications to the relevant Registry keys were executed by `services.exe`, not `sc.exe`, with no evident connection between the two processes. Further investigation and reference to the Windows Developer Reference [182] revealed that these processes communicate via ALPC, a method of inter-process communication (IPC) in Windows that is not usually recorded by standard logging frameworks. Additional logging through Windows ETW "NT Kernel Logger" validated this connection, although it produced excessively large datasets due to the extensive utilization of ALPC on Windows.

To address this, we introduce the notion of *expert-guided edges* in CPD and define them as follows:

**Definition 2** *Expert-guided edges are edges formed by applying specialized parsing rules during log processing for provenance graph generation. This method embeds expert knowledge about process creation routines and operating system policies into the backward and forward tracing process.*

For instance, it is possible to associate `sc.exe` with `services.exe` when `services.exe` alters a Registry key at the designated location immediately following the execution of `sc.exe` using the identical service name, as demonstrated in Figure 7.3. This method provides three advantages: an expedited search

[1] `sc.exe` is a typical example of Living-Off-the-Land Binaries (Section 2.2).

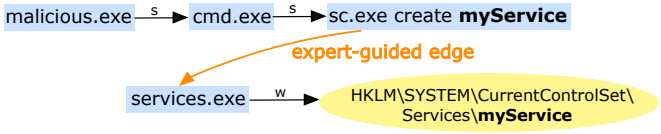



**Figure 7.3:** An expert-guided edge created during reconstruction of a T1543.003 persistence setup attack graph [P4]. An attacker-controlled malicious process leverages LOLBins to create a malicious service for persistence. The indicative Registry key is however modified by a Windows system process, to which no link from the malicious process can be built using logs from standard logging frameworks.

**Algorithm 2:** Expert-guided Edge Creation [P4]

**Inputs :** System audit log events $\mathcal{E}$;
List of critical system processes $L_{<\mathcal{P}>}$
**Output :** List of dependency path $L_{<P>}$

**foreach** $\mathcal{E}_\kappa \in \mathcal{E}$ **do**
  /* Get the process of current event */
  $\mathcal{P}_\kappa \leftarrow$ GetSubject($\mathcal{E}_\kappa$)
  **if** $\mathcal{P}_\kappa \in L_{<\mathcal{P}>}$ **then**
    /* Add dependency path to the standard routine nodes */
    $P \leftarrow$ AddPathToRoutineNodes($\mathcal{P}_\kappa$)
    $L_{<P>} \leftarrow L_{<P>} \cup P$
  **end**
**end**
**return** $L_P$

process, a decrease in dependency proliferation, and the ability to close gaps that would otherwise remain unbridgeable.

Likewise, we identify gaps in the Linux environment through the analysis of Auditd logs, despite our comprehensive monitoring of a wide array of syscalls. Notably, during the persistence setup for T1543.002 (Systemd Service) [121], an indicative file `/etc/systemd/system/*.service` is generated. However, during the persistence execution, we cannot observe access to the aforementioned file; instead, we find a related in-memory file at `/sys/fs/cgroup/system.slice/*.service/*` that shares the same service file name. This situation prevents the establishment of an edge between the relevant subgraphs when adhering to the conventional principle of "writing and reading on the same file node." To address this challenge, we implement expert-guided edges. The process for creating these expert-guided edges is outlined in Algorithm 2, and is utilized in Line 31 of Algorithm 1. It is important to note that

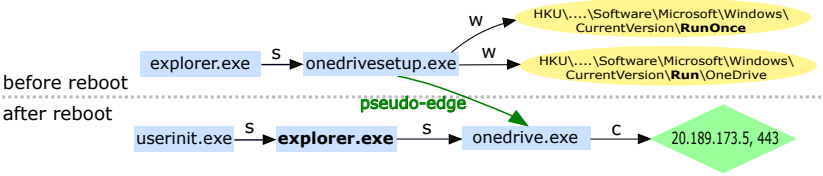



**Figure 7.4:** A false-positive persistence attack graph automatically generated by CPD on the AVIATOR-APT29-1 dataset [P4]. This graph wrongly classifies an instance of T1547.001. It turns out to be a benign program, i.e., Microsoft OneDrive, leveraging Registry Run keys for updates. It in fact connects back to an IP address belonging to Microsoft Corporation.

the creation of expert-guided edges is predicated on the assumption that the operating system's integrity remains intact. The reduction rate in log storage achieved through the introduction of expert-guided edges is presented in Section 7.4.2.

### 7.2.3 False Positive Reduction

The task of constructing an pseudo-edge that accurately reflects a persistence attack is challenging. The difficulty primarily arises from the fact that numerous legitimate applications utilize Registry Run keys (T1547.001) [132], Windows services (T1543.003) [121], scheduled tasks (T1053.005) [183] etc. for some program-specific routines involving remote connections. Consequently, this results in a high incidence of false-positive persistence attack graphs. For example, certain benign applications, such as Adobe Acrobat's Update Service (`armsvc.exe`), establish a Windows service to periodically check for and download updates, thereby imitating persistence behaviors and generating false-positive pseudo-edges in CPD. While the act of creating a Windows service resembles the persistence setup, the subsequent update downloads, which involve remote connections, further mimic persistence execution. Likewise, applications like Google Chrome and Microsoft OneDrive employ Registry Run keys for their update processes, often resulting in erroneous alerts. Figure 7.4 illustrates a typical false-positive persistence attack graph produced by CPD following its second stage.

In response to this issue, we have created an algorithm aimed at reducing false positives by employing contextual indicators to distinguish between

benign and malicious activities. This algorithm incorporates the concept of *pseudo-edge strength* and ranks pseudo-edges according to a computed threat score. We classify pseudo-edges into two categories: causality-based and correlation-based, enhancing detection accuracy by acknowledging the distinct characteristics of various persistence techniques. Correlation-based pseudo-edges, which pertain to login account methods such as T1098 (account manipulation) [184], T1136 (create account) [185], and T1078 (valid accounts) [133], exhibit lower reliability *due to the uncertainty of user identity behind consecutive logins*. That is, comparing to causality-based pseudo-edges, which are assigned to the rest of persistence techniques, correlation-based pseudo-edges are inevitably given another degree of inconfidence in persistence detection. Because we can hardly be sure that two consecutive logins (with the same account) into one machine are performed by the same user / APT actor, if they both hold the credentials. Consequently, these are assigned a 'penalty' weight (less than 1) in the anomaly score calculation (Equation 7.3). For accurate detection, we leverage context from the Cyber Kill Chain tactic and technique levels, as well as from the program execution level.

#### 7.2.3.1 Causality-based pseudo-edges

We formulate the following indicators based on studying APT behaviors in real-world attacks. Notice that the "causality" in causality-based pseudo-edges applies to the relationship between a persistence setup sub-graph and the corresponding persistence execution sub-graph.

- **Degree of indirection** in both persistence setup and persistence execution. Analyzing the real-world activities of APT29 reveals that multiple layers of indirection are employed to initiate the Command and Control (C2) agent program. Specifically, when the malicious shortcut file located in a Windows startup folder is accessed, the process `explorer.exe` executes a seemingly benign batch file associated with the shortcut file (indirection 1). Subsequently, the resulted `cmd.exe` process launches a `powershell.exe` process as instructed by the batch file (indirection 2). This `powershell.exe` process then initiates another process called `hostui.exe` that appears normal (indirection 3). Following this, another `powershell.exe` process is started by the normal-looking process (indirection 4), which leads to the initiation of yet another `powershell.exe` process (indirection 5) prior to es-

tablishing a connection with the C2 server. This sequence clearly diverges from the typical usage of Windows startup folders by legitimate programs.

- **Credential access tactic** is frequently undertaken as a means to obtain the "low-hanging fruits" persistence, as observed in APT groups' past behaviors, e.g., APT29 [186], Sandworm [128], and Wizard Spider [187].
- **Persistence techniques are often executed in tandem**, as, together, they are likely to foster a greater degree of reliability in persistence.
- APT actors typically implement persistence techniques **immediately following initial access or lateral movement**. For instance, Wizard Spider [187] established persistence on the first compromised victim machine and subsequently on the second victim machine after executing lateral movement.
- It is common for APT actors to utilize certain **discovery techniques before** they engage in persistence techniques. However, our analysis indicates that this indicator is generally less reliable, and exhibits a higher level of "noise" than other indicators. To address this concern, we assign the smallest weighting factor to this indicator during the computation of the anomaly score, as specified in Equation 7.3.

#### 7.2.3.2 Correlation-based pseudo-edges

Furthermore, we divide correlation-based pseudo-edges into two separate categories.

*Type 1 - persistent initial re-access* We have discerned the indicators listed below that pertain to this category of persistence.

- The observation of **credential access tactic** serves as a significant indicator for correlation-based pseudo-edges as well. For example, OS credential dumping (T1003) [188] is frequently executed on compromised systems, with the acquired credentials subsequently utilized for remote reconnection. Additionally, the **intensity of usage**, which refers to the frequency of the same technique, along with the **extensiveness**, indicating the variety of attempted credential access techniques, can serve as a weighting factor. A case in point is one of the APT group Sandworm's operations, in which they deployed an executable on the target machine to extract web credentials,

alongside another executable designed for keylogging a legitimate user's RDP session to capture domain credentials.

- The computer attempting to access the system lacks a **valid Fully Qualified Domain Name (FQDN)** or a **registered computer account** within the domain. This situation is indicative of a potentially malicious attempt at re-access, as attackers generally do not possess physical ownership of a computer that is joined to the domain.
- **Local account creation** following lateral movement, such as during a WinRM [189] session.
- **Installing, activating and enabling standard remote access tools** such as VNC and RDP server. For instance, Carbank [190] installed a VNC server on its compromised machine to ensure persistence following the acquisition of credentials, and subsequently opened the corresponding port on the firewall.

*Type 2 - persistent lateral movement* Type 2 correlation-based pseudo-edges represent a special kind of pseudo-edges that signify the intersection of persistence tactic and lateral movement tactic. The subsequent indicators are derived from an analysis of real-world APT attacks observed in the past.

- **Remote system discovery** (T1018) [191] occurs on a domain-joined computer prior to establishing a remote connection to another computer within the network. For example, the APT group APT29 utilized LDAP (Lightweight Directory Access Protocol) queries to identify additional hosts in the domain before initiating a remote PowerShell session in a secondary target.
- **Ingress tool transfer** (T1105) [192] is executed from one domain-joined computer to another prior to the establishment of a remote connection.
- **Credential access tactic** is employed on a domain-joined computer before initiating a remote connection. For instance, the APT group Sandworm has acquired SSH keys from the initially compromised computer and subsequently leveraged these credentials to facilitate lateral movement to a second machine.
- **Local account creation** during or following lateral movement activities.
- **Installing, activating and enabling standard remote access tools** such as VNC and RDP server.

Considering the aforementioned indicators, CPD computes an anomaly/threat score, which is utilized to assess the strength of pseudo-edges and to prioritize them accordingly for investigation. This approach ensures that the most probable malicious pseudo-edges are examined first by a security analyst. In the concluding phase of CPD, as delineated in Algorithm 3, each pseudo-edge is initially categorized into one of the three specified classifications (Line 4). Subsequently, the corresponding persistence attack graph undergoes an automatic analysis to extract features associated with the previously mentioned indicators, such as the degree of indirection during both the setup and execution of persistence. Additionally, the graph is further scrutinized to incorporate more contextual information and to identify the presence of other indicators, such as related attack steps within a Cyber Kill Chain, e.g., whether credential access is detected in its dependency graph. The dependency graph builds upon the persistence attack graph, thus providing a more detailed representation. Consequently, our false positive reduction algorithm utilizes all indicators identified in both the concise persistence attack graph and its more detailed dependency graph. The subsequent section will elucidate how these indicators are employed for the assignment of anomaly scores through three equations.

First, we determine the anomaly score of an indicator that can be observed from the persistence attack graph in the following manner (Line 8 in Algorithm 3):

$$AS_{ind-PAG} = N_s^2 \times N_e^2 \tag{7.1}$$

where $N_s$ and $N_e$ denote the degree of indirection in persistence setup atomic graph and persistence execution atomic graph, respectively.

Second, the calculation of the anomaly score for an indicator observable from the dependency graph is conducted as follows (Line 19 in Algorithm 3):

$$AS_{ind-DG} = \begin{cases} \max\limits_{i}\left(\frac{D_c}{D_s} \times Freq(teq_i) \times Var(tac)\right) & t_{teq} \leq t_e \\ \max\limits_{i}\left(\frac{D_c}{D_e} \times Freq(teq_i) \times Var(tac)\right) & t_{teq} > t_e \end{cases} \tag{7.2}$$

**Algorithm 3:** FALSE POSITIVE REDUCTION [P4]

**Inputs :** System audit log events $\mathcal{E}$;
List $L_{<PE,PAG>}$ of pseudo-edge and persistence attack graph pairs;
List $L_{<ind-PAG>}$ of indicators inside persistence attack graphs;
List $L_{<ind-DG>}$ of indicators inside dependency graphs;
Max persistence-edge alert number $\mathcal{N}$

**Output :** List $L_{<PE,AS>}$ of persistence edge and its anomaly score pairs

**foreach** $< PE(\gamma,\alpha), PAG(\gamma,\alpha) > \in L_{<PE,PAG>}$ **do**
  $AS_{PE(\gamma,\alpha)} \leftarrow 0$
  $L_{<AS_{PE}>} \leftarrow 0$
  /* Classify pseudo-edge */
  $PE'(\gamma,\alpha) \leftarrow$ GETCATEGORY$(PE(\gamma,\alpha))$
  /* Select indicators based on pseudo-edge type */
  $L'_{<ind-PAG>} \leftarrow$ GETINDICATORS$(PE'(\gamma,\alpha), L_{<ind-PAG>})$
  $L'_{<ind-DG>} \leftarrow$ GETINDICATORS$(PE'(\gamma,\alpha), L_{<ind-DG>})$
  **foreach** $indicator \in L'_{<ind-PAG>}$ **do**
    $AS_{indicator} \leftarrow$ CALCULATESCORE1$(indicator, PAG(\gamma,\alpha))$
    $L_{<AS_{PE}>} \leftarrow L_{<AS_{PE}>} \cup AS_{indicator}$
  **end**
  $(L_{<\mathcal{E}\delta>}, DG(\delta)) \leftarrow$ TRAVERSALBACKWARD$(PAG(\gamma,\alpha), \mathcal{E})$
  $(L_{<\mathcal{E}\eta>}, DG(\eta)) \leftarrow$ TRAVERSALFORWARD$(PAG(\gamma,\alpha), \mathcal{E})$
  $DG(\kappa) \leftarrow$ MERGEGRAPH$(DG(\delta), DG(\eta))$
  $L_{<\mathcal{E}\kappa>} \leftarrow L_{<\mathcal{E}\delta>} \cup L_{<\mathcal{E}\eta>}$
  **foreach** $\mathcal{E}_\kappa \in L_{<\mathcal{E}\kappa>}$ **do**
    **foreach** $indicator \in L'_{<ind-DG>}$ **do**
      $satisfied \leftarrow$ CHECKINDICATOR$(indicator, \mathcal{E}_\kappa)$
      **if** $satisfied$ **then**
        $AS_{indicator} \leftarrow$ CALCULATESCORE2$(indicator, DG(\kappa))$
        $L_{<AS_{PE}>} \leftarrow L_{<AS_{PE}>} \cup AS_{indicator}$
      **end**
    **end**
  **end**
  $AS_{PE(\gamma,\alpha)} \leftarrow$ SUMSCORE$(L_{<AS_{PE}>})$
  $L_{<PE,AS>} \leftarrow L_{<PE,AS>} \cup AS_{PE(\gamma,\alpha)}$
**end**
$L_{<PE,AS>} \leftarrow$ SORTBYSCORE$(L_{<PE,AS>})$
$L_{<PE,AS>} \leftarrow$ REMOVEBYBUDGET$(L_{<PE,AS>}, \mathcal{N})$
**return** $L_{<PE,AS>}$

where the variable $D_s$ represents the distance between an indicative attack step, such as the dumping of OS credentials, and the step involved in persistence setup. This distance is measured as the number of hops between the corresponding two processes. In a similar manner, $D_e$ indicates the distance between the persistence execution step and an indicative attack step, for instance, remote system discovery. The variable $D_c$ is a predetermined cut-off value that signifies the maximum number of hops deemed to have a positive influence on the anomaly score. This mechanism serves to penalize an indicative attack step that is excessively distant from either the persistence setup or execution step, resulting in a weighting factor in Equation 7.2 being less than 1 when $D_s$ or $D_e$ exceeds $D_c$. The variable $Freq(teq)$ denotes the frequency of the same attack technique being executed as a repeated attempt, while $Var(tac)$ reflects the breadth of usage of the same tactic, specifically the number of distinct techniques employed within that tactic. This rationale is based on the understanding that attackers frequently utilize a range of techniques from the same tactic to enhance their likelihood of success. The variable $t_{teq}$ indicates the time at which an attack step occurs, whereas $t_e$ marks the time of persistence execution. In cases where multiple attack techniques are identified for the same indicator, only the one with the highest anomaly score is taken into account.

Ultimately, the final anomaly score is derived from Equation 7.3 (Line 24 in Algorithm 3).

$$AS_{PE} = \prod_{i=1}^{n} (AS_i)^{w_i} \tag{7.3}$$

where $n$ represents the total number of identified indicators associated with a specific pseudo-edge, while $AS_i$ denotes the anomaly score of an indicator derived from either Equation 7.1 or Equation 7.2 pertinent to this pseudo-edge. The variable $w_i$ serves as a weighting factor. Subsequently, the pair consisting of the pseudo-edge and its corresponding anomaly score is added into a list (as indicated in Line 25 of Algorithm 3), which is at the end sorted by anomaly score (as shown in Line 27 of Algorithm 3). pseudo-edges that are ranked below the $\mathcal{N}$-th pseudo-edge are classified as false-positive pseudo-edges and are thus eliminated from the list (Line 28 in Algorithm 3). The final output of Algorithm 3 is the refined list of pseudo-edge and anomaly score pairs.

## 7.3 Implementation

We developed a prototype of CPD in Python and deployed it on a 64-bit Ubuntu 23.04 operating system equipped with 512 GB of RAM and a 64-core AMD processor. This machine hosts multiple virtual machines utilized by various researchers to conduct independent scientific experiments concurrently. Our implementation interfaces Elasticsearch [169] through EQL, a query language tailored for security applications. Elasticsearch facilitates scalable and near real-time search capabilities for log data analysis. Additionally, we employ a Python library called NetworkX [176] to generate provenance graphs on demand, and utilize another Python library called PyVis [177] for the visualization of these graphs.

For the purpose of log collection on Linux systems, we employ Auditd [85]. In contrast, our primary tool for Windows is System Monitor (Sysmon) [84]. Sysmon distinguishes itself from Windows ETW [83] by generating a process GUID for each process, which significantly reduces the occurrence of false dependencies in post-processing. Nevertheless, Sysmon does not provide the capability to collect file and Registry read operations; thus, we utilize Windows Security [155] audit logs to capture all file and Registry interactions. Furthermore, we also collect ALPC logs via the "NT Kernel Logger" [163] ETW session

## 7.4 Evaluation

In this section, we assess the performance and efficiency of CPD as a dedicated system for detecting persistence.

**Public Datasets.** Public datasets frequently exhibit a deficiency in persistence traces. From the available DARPA datasets, we selected the DARPA-E5 dataset [70] and the DARPA-OpTC dataset [71], while excluding the DARPA-E3 [69] due to its absence of persistence attacks. The DARPA-E5 dataset encompasses emulated APT attacks, with the Fivedirections subset, which is centered on Windows, containing two instances of persistence attacks. We evaluated only this subset with CPD. The DARPA-OpTC dataset presents a total of three instances of persistence; however, only one of these instances fulfills all the criteria for effective persistence as outlined in Section 4.2. The

**Table 7.2:** Overview of Cpd's evaluation datasets [P4].

| Dataset | Target Host Number | Persistence Attack Number | Target Host OS | Data Size | Event Number |
|---|---|---|---|---|---|
| **ATLASv2** | 2 | 0 | Windows | 26GB | 5.6M |
| **DARPA-E5-Fivediretions** | 3 | 2 | Windows | 348GB | 1.4B |
| **DARPA-OpTC** | 50 (/500)[a] | 1 | Windows | 380GB | 338M |
| **Aviator-APT29-1** | 3 | 2 | Windows | 32GB | 22M |
| **Aviator-APT29-2** | 3 | 3 | Windows | 24GB | 14M |
| **Aviator-Sandworm-1** | 4 | 4 | Windows Linux | 68GB | 57M |

[a] The DARPA-OpTC dataset contains logs from 500 Windows machines. We evaluated Cpd on a subset of 50 machines, including the 3 machines with persistence and another 47 random machines.

remaining two instances did not satisfy the third condition, as the attacks concluded prematurely. Additionally, we incorporated the ATLASv2 dataset [193] due to its Carbon Black Cloud (CBC)) [194] detection results on persistence. The ATLASv2 dataset provides a greater volume of background activities and more comprehensive logging compared to the ATLAS dataset [32], utilizing tools such as Sysmon and VMware CBC. Although it does not contain actual persistence attacks, it does include persistence alerts created by VMware CBC, which we utilized for comparative analysis with Cpd. The detection results obtained from Cpd on these datasets were validated against the provided ground truth.

**MITRE Attack Emulation.** MITRE has developed eleven comprehensive emulation plans [46], which are grounded in actual APT behaviors and each incorporates a minimum of two persistence techniques. These plans are utilized in the MITRE Engenuity ATT&CK® Evaluations [147] to evaluate commercial EDR (Endpoint Detection and Response) systems. However, MITRE has not released any associated datasets. We meticulously executed two emulation plans, focusing on the ten most "persistent" APT groups discussed in Section 4.2, and subsequently assessed Cpd using these datasets. The emulation plan for APT29 features two unique scenarios, whereas the two scenarios for Sandworm are the same. We generated three datasets by emulating APT29 scenario 1, APT29 scenario 2, and Sandworm scenario 1,

which we designated as AVIATOR-APT29-1, AVIATOR-APT29-2, and AVIATOR-Sandworm-1, respectively. These three datasets are incorporated as subsets within the AVIATOR dataset. An overview of our datasets is provided in Table 7.2.

### 7.4.1 Effectiveness of CPD

#### 7.4.1.1 CPD vs. SIEM detection rules

Upon completing the development of our persistence detection rule set, we conducted a comparative analysis with widely recognized open-source SIEM detection rules from Elastic [146], Sigma [74] and Google Chronicle [75]. We extracted all detection rules pertinent to persistence from these sources, transformed them into EQL queries, and executed them alongside our system on datasets that included persistence attack scenarios. The findings presented in Table 7.3 indicate that the open-source SIEM rules resulted in a higher number of false positives and failed to identify true attacks, whereas our system CPD demonstrated a marked improvement over these rules. A further analysis of the second and third stages of our system, and the comparison between them, are illustrated in Figure 7.5, which shows the cumulative distribution of threat scores for both benign and attack pseudo-edges. The results for the stage three, as detailed in Table 7.3, are derived from the lowest threshold of true attack threat scores. Our system CPD's final stage significantly minimized false positives, reducing them from thousands in the open-source rules to merely dozens per dataset. Moreover, the false positive reduction algorithm implemented in the stage three of CPD achieved an average reduction rate of 83%, thereby greatly improving overall accuracy.

**Table 7.3:** Comparison of CPD and open-source SIEM rules [P4].

| Dataset | TP[a] | CPD | | | | | | Elastic | | Chronicle | | Sigma | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Stage 1[b] | | Stage 2 | | Stage 3 | | | | | | | |
| | | FP | FN | FP | FN | FP | FN | FP | FN | FP | FN | FP | FN |
| DARPA-E5 | 2 | 21911 | 0 | 117 | 0 | 7 | 0 | 5371 | 1 | -[c] | - | 57869 | 0 |
| DARPA-OpTC | 1 | 63460 | 0 | 35 | 0 | 4 | 0 | 15111 | 1 | 47584 | 0 | 11567 | 0 |
| AVIATOR-APT29-1 | 2 | 4489 | 0 | 82 | 0 | 23 | 0 | 260 | 2 | 1680 | 1 | 15158 | 0 |
| AVIATOR-APT29-2 | 3 | 3256 | 0 | 62 | 0 | 14 | 0 | 351 | 2 | 1724 | 1 | 13305 | 0 |
| AVIATOR-Sandworm-1 | 4 | 3881 | 0 | 48 | 0 | 8 | 0 | 527 | 1 | 1209 | 0 | 63760 | 0 |

[a] TP = True Positives, which represents the number of true persistence attack instances in the corresponding dataset.

[b] CPD's stage 1 result refers to the detection result after only applying our persistence setup detection rules, while CPD's stage 2 result refers to the detection result after also checking persistence execution and then incorporating pseudo-edges, and CPD's stage 3 result refers to the detection result after further applying our false positive reduction algorithm.

[c] Chronicle's rules overly use process names or paths as conditions. However, in the DARPA-E5 dataset, process name is not present in log events that are caused by processes created before the logging framework started. Lacking this information would result in wrong results. Thus, we do not evaluate Chronicle's rules on the DARPA-E5 dataset.

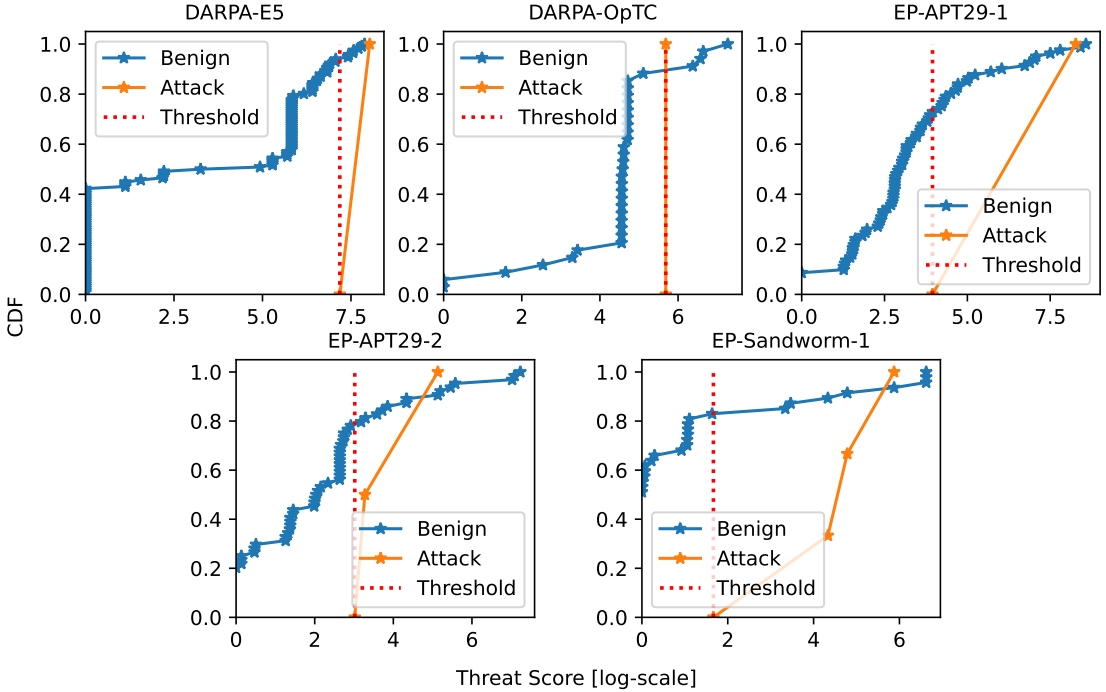



**Figure 7.5:** CDF of threat score for false and true alerts [P4].

The significant enhancement of CPD over traditional SIEM detection rules is primarily attributed to two fundamental insights, which are implemented in stages two and three of CPD. The first critical insight is that all persistence attacks necessitate an initial setup phase that exploits a "sensitive" system functionality, followed by an execution phase that involves remote connections. While both SIEM rules and CPD's stage one's rules trigger alerts when persistence-related system functionalities are engaged, they do not specifically address instances of misuse. In contrast to SIEM rules, which fail to correlate log events occurring at different times, CPD's stage two evaluates whether a remote connection can be traced back to a persistence setup alert. This approach effectively eliminates alerts related to system activities that are benign in nature. The second important insight is that persistence represents merely one of several stages within a Cyber Kill Chain that must be executed in conjunction to fulfill the objectives of attackers. This crucial insight is elaborated upon in our false positive reduction algorithm detailed in Section 7.2.3. Both insights fully utilize the capabilities of provenance analytics, which provide context through the examination of system activities over an extended timeframe.

All parameters within our algorithm designed for false positive reduction, such as weighting factors, remain consistent across different datasets. Similar to earlier studies [24]–[26], our objective is not to eliminate all false positives but to emphasize the most probable attacks for further examination through threat score ranking. In essence, CPD seeks to enhance the probability of identifying persistence attacks while considering the constraints of available human resources. Security analysts can adjust this parameter according to their alert budget.

We conduct a thorough examination of the reasons behind the undesirable outcomes associated with the open-source detection rules, specifically addressing two key issues: 1) the prevalence of false negatives (FN) when utilizing Elastic's rules; and 2) the exceptionally high rate of false positives generated by Sigma's rules. To address the first issue, we meticulously analyze Elastic's persistence detection rules. Our findings indicate that these rules not only excessively allowlist certain programs but also exhibit a tendency to be overly specific, incorporating numerous hard-coded strings as conditions. These factors render the rules particularly susceptible to evasion attacks, resulting in a significant number of false negatives in practical applications. For example, one of Elastic's rules pertaining to T1547.001 [195] allows all programs located not only in `C:\Windows\system32\` but also in `C:\ProgramFiles\`, effectively

excluding a vast majority of programs on the Windows operating system. As previously mentioned, the DARPA-OpTC dataset encompasses three persistence instances, two of which are classified as "unsuccessful" solely due to incomplete attack scenarios. Our analysis reveals that the high degree of specificity and excessive allowlisting in Elastic's rules led to the omission of all three persistence instances. In contrast, our system CPD successfully detected all three persistence instances during the initial stage and eliminated the two unsuccessful instances in the subsequent stage, ultimately presenting only the genuine persistence as the final output.

To address the second question, we examine Sigma's persistence detection rules through manual inspection. Our analysis reveals that Sigma's rules exhibit a wide range of specificity, with some being excessively detailed while others are overly broad. For instance, a sample Sigma rule for T1574.009 [196] relies solely on a program file path as its condition, resulting in an excessive number of alerts. Additionally, we observe that the high alert volume is partly attributable to the presence of numerous similar rules within the Sigma rule repository, which have been created by various contributors for the same attack techniques. We argue that the maintainers of the repository should enhance the organization of the rules and eliminate those that appear redundant.

In contrast to Sigma, Chronicle's rules generate fewer alerts, although they do present some false negatives. Furthermore, we note that Chronicle has some overly simplistic rules that lead to an excessive number of alerts, such as the rule for T1053.005 [197]. It is important to highlight that this study concentrates on the detection of persistence, a topic that is frequently neglected or poorly understood, as discussed in Section 4.2. Therefore, it is not surprising that these widely used detection rule repositories lack robust or comprehensive persistence detection rule sets.

#### 7.4.1.2 CPD vs. VMware CBC

We subsequently conduct a comparison between our system CPD and a commercial Endpoint Detection and Response (EDR) solution, i.e., VMware CBC [194], on the ATLASv2 dataset. The detection outcomes for both CPD and VMware CBC are presented in Table 7.4. Like above, even without executing its third stage, CPD surpasses VMware CBC by achieving a reduction in the false positive rate of 80%. Furthermore, our analysis reveals that the

**Table 7.4:** Comparison of CPD and CBC EDR [P4].

| Dataset | CPD | | | | | | CBC | |
|---|---|---|---|---|---|---|---|---|
| | Stage 1 | | Stage 2 | | Stage 3 | | | |
| | FP | FN | FP | FN | FP | FN | FP | FN |
| ATLASv2 | 1602 | 0 | 11 | 0 | -[a] | - | 56 | 0 |

[a] As the ATLASv2 dataset does not include a real persistence attack, for fair comparison, we did not run CPD's stage 3 on the this dataset.

indicators of compromise (IOC) from VMware CBC demonstrate a tendency to indiscriminately allowlist certain programs, similar to the SIEM detection rules mentioned earlier. This practice can easily lead to false negatives.

It is important to emphasize that the first stage of CPD effectively mitigates excessive false alarms that arise from critical system programs altering files and Registry entries at indicative locations. However, CPD first verifies whether these allowlisted programs may be compromised by checking for modifications to their executable files. To enhance the detection outcomes in the first stage of CPD, it refrains from generating unnecessary alerts for DLL (Dynamic-Link Library) files that are temporarily placed on disk and subsequently deleted. Failing to do so would result in numerous security alerts associated with various persistence techniques, such as T1574.001 [198] and T1574.002 [199]. This behavior is typical of Windows programs, which often drop DLL files into a temporary folder upon initiation, subsequently launching new instances (as child processes) that utilize these DLL files. The DLL files are removed when the child processes terminate. However, in contrast to Linux, Windows does not automatically delete temporary files, leaving this task to the programs themselves. Consequently, temporary files may persist through reboots if not removed by their creators. Therefore, if a DLL file is dropped and remains undeleted, CPD will issue a persistence setup alert for it during its first stage.

#### 7.4.1.3 CPD vs. prior PIDS

Most state-of-the-art heuristics-based PIDS [24]–[26] have been assessed using either proprietary datasets or datasets that do not include persistence

attacks. Consequently, a direct comparison with these systems is not feasible. By their very design, these systems are likely to struggle with detecting persistence. This is due to the fact that if attackers decompose the entire Cyber Kill Chain into multiple phases, as illustrated in Figure 4.1, neither forward nor backward tracing will successfully identify an event of interest in subsequent phases. In reality, persistence attacks can be employed to completely circumvent these detection systems.

Therefore, we envisioned a comparison with three of the most recent, state-of-the-art learning- or anomaly-based PIDS: KAIROS [35], FLASH [36], and MAGIC [115]. These systems demonstrate superior performance compared to other learning-based PIDS such as [28], [33], [81], [114], [137]. However, none of these studies provide detection results specifically related to persistence attacks across their respective datasets. The attack graphs presented in the original publications were derived from subsets of the DARPA-E3, DARPA-E5, or DARPA-OpTC datasets, none of which include genuine persistence attacks. This is little surprising, given our earlier discussion regarding the frequent absence of persistence traces in public datasets. Authentic persistence attacks are only found in the DARPA-E5 Fivedirections subset and the DARPA-OpTC day 2 subset.

While MAGIC has been evaluated using the DARPA-E3 dataset, it has not been tested on the more recent DARPA-E5 dataset, DARPA-OpTC dataset, or any datasets that contain true persistence attacks. Both KAIROS and FLASH have been assessed using the DARPA-OpTC dataset. Although KAIROS has also been evaluated on several subsets of DARPA-E5, it does not include the Fivedirections subset. Furthermore, KAIROS does not provide the necessary information to operate on the DARPA-E5 Fivedirections subset. Given that both KAIROS and FLASH are complex systems with numerous hyperparameters and model configurations, ensuring a fair extension of their evaluation to the DARPA-E5 Fivedirections subset poses significant challenges. Therefore, we utilized the pre-trained model weights for the DARPA-OpTC dataset.

Both KAIROS [35] and FLASH [36] exhibit enhanced attack detection capabilities at a more granular level compared to earlier PIDS such as UNICORN [28]. KAIROS takes system events as input, dividing the entire timeline into multiple time windows and categorizing each as either benign or malicious. In contrast, FLASH can classify each node in the provenance graph as benign or malicious. The detection results of KAIROS on the DARPA-OpTC dataset

**Table 7.5:** Comparison of CPD, KAIROS and FLASH on DARPA-OpTC (Host 0501) [P4]. ✓= Detected, ✗= Not Detected

| | Persistence Setup | Persistence Execution | Run Time (minute)[a] | Mean Memory Consumption (GB) |
|---|---|---|---|---|
| **CPD** | ✓ | ✓ | 2 | 2.6 |
| **KAIROS [35]** | ✓ | ✗ | 630 | 10.2 |
| **FLASH [36]** | ✓ | ✗ | 312 | 9.1 |

[a] From data processing (including training) to detection result.

indicate that it accurately identified the time window associated with persistence setup as malicious, yet incorrectly classified the time window for the corresponding persistence execution as benign. Similarly, FLASH identifies a collection of malicious nodes, which includes those linked to persistence setup, but omits the nodes that are accountable for persistence execution. As illustrated in Table 7.5, CPD, designed specifically for persistence detection, utilizes four times less memory than both KAIROS and FLASH, while demonstrating greater accuracy and operating 315 times faster than KAIROS and 156 times faster than FLASH. Note that we include the time for training in the run time as well. Our system CPD does not need training. Besides, CPD integrates with Elasticsearch, a scalable and nearly real-time search engine, for rule matching and conducts provenance analytics solely on system events pertinent to persistence setup and execution.

### 7.4.2 Log Reduction via Expert-Guided Edges

As outlined in Section 7.2, the system logs produced by the most widely used logging frameworks reveal gaps due to the lack of IPC log events. Nevertheless, by incorporating expert-guided edges, we can reduce reliance on IPC events while still effectively identifying pertinent attack steps. In the course of implementing the full emulation plans from MITRE, we gather ALPC log events through the "NT Kernel Logger" [163] ETW trace session. The log reduction rate of CPD, achieved by utilizing expert-guided edges instead of depending on ALPC logs across each dataset, is presented in Table 7.6. It shows that CPD has a log reduction rate of 20% to 53%.

**Table 7.6:** Log reduction rate of expert-guided edges [P4].

| Dataset | Data Size | ALPC Data Size | Reduction Rate |
|---|---|---|---|
| Aviator-APT29-1 | 32GB | 12GB | 38% |
| Aviator-APT29-2 | 24GB | 5GB | 20% |
| Aviator-Sandworm-1 | 68GB | 36GB | 53% |

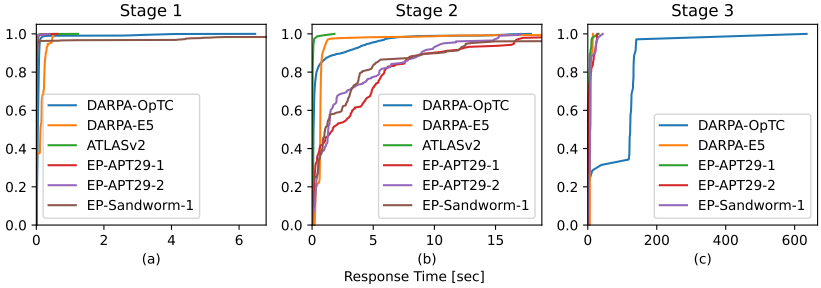



**Figure 7.6:** CDF of response time of Cpd [P4].

### 7.4.3 Response Time & Runtime Overhead

The response time of Cpd is divided into three distinct parts, and is measured *per persistence attack graph*, i.e., the time it takes to output a single persistence attack graph with calculated threat score. The cumulative distribution function of Cpd's response time across its three stages is illustrated in Figure 7.6. The first stage's response time is measured per detection rule matching. As depicted in Figure 7.6 (a), it takes less than half a second to find alert events in an entire dataset for more than 95% of detection rules. The stage two's response time is measured as the time to perform backward tracing on an indicative process with remote connections, while matching against persistence execution rules, aligning persistence tables, and creating pseudo-edge. Figure 7.6 (b) indicates that it takes less than 11 seconds to generate a pseudo-edge for more than 90% of all indicative processes. The third stage's response time is the time taken to conduct both backward and forward tracing on each alert generated from the second stage, while also assessing contextual indicators and calculating the threat score. As shown in Figure 7.6 (c), inspecting each alert takes less than 140 seconds for over 95%

**Table 7.7:** Memory utilization (MB) of running CPD [P4].

| | DARPA-OpTC | DARPA-E5 | ATLASv2 | AVIATOR-APT29-1 | AVIATOR-APT29-2 | AVIATOR-Sandworm-1 |
|---|---|---|---|---|---|---|
| max | 10830 | 18464 | 579 | 3625 | 3513 | 3484 |
| mean | 4484 | 11720 | 247 | 892 | 895 | 573 |

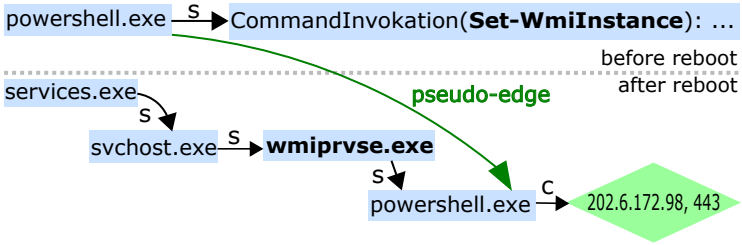



**Figure 7.7:** WMI persistence attack graph automatically generated by CPD on the DARPA-OpTC dataset [P4].

of alerts in the DARPA-OpTC dataset, and under 42 seconds for all alerts in other datasets. The memory overhead incurred by executing CPD on each dataset is detailed in Table 7.7. These results were obtained using `mprof` [200], which samples memory usage every 100 milliseconds. It shows that CPD's memory consumption largely depends on the datasets.

### 7.4.4 Reconstructed Persistence Attack Graphs

A reconstructed persistence attack graph derived from the public dataset DARPA-OpTC is illustrated in Figure 7.7. This figure represents a true positive for T1546.003 (Event Triggered Execution: WMI Event Subscription) [201]. Similar to Figures 7.2 and 7.4 discussed in Section 7.2, the upper section of the figure displays the persistence setup sub-graph, while the lower section presents the persistence execution sub-graph. These two sub-graphs are linked by an pseudo-edge. This concise attack graph demonstrates that the attacker created a WMI (Windows Management Instrumentation) instance during persistence setup. In the persistence execution phase, the Powershell code responsible for reconnecting to the attacker is executed through the Windows system process `wmiprvse.exe` upon the triggering of a specific

event. It is important to emphasize that the accuracy of this concise attack graph, which encapsulates the most vital information regarding persistence, can significantly assist even novice security analysts in conducting a swift and comprehensive attack investigation.

## 7.5 Limitations & Discussion

**Mimicry Attacks.** CPD is not designed to identify all persistence techniques. While it has not been evaluated for detecting macOS-based persistence techniques, we believe that the underlying principles can be applied. However, identifying persistence techniques related to cloud infrastructures may necessitate a distinct approach that takes into account the specific conditions of cloud environments. Similar to other provenance-based detection systems that operate under the assumption of OS integrity, CPD is unable to detect pre-OS boot persistence attacks, such as Bootkit. Nevertheless, the persistence techniques that remain unaddressed do not belong to the most commonly exploited top 10 persistence techniques, suggesting that this limitation will have a minimal effect on the overall functionality of CPD. Furthermore, CPD demonstrates resilience against evasion techniques outlined in recent studies, as these techniques primarily target anomaly-based PIDS that rely on path-based or graph-based embedding. Heuristic-based systems like ours are inherently more challenging to evade, as the introduction of additional "camouflaging" events has a negligible impact on the detection outcomes of these systems.

**Maintain and Extend CPD.** One constraint of CPD is its dependence on the existing threat intelligence knowledge base, specifically the MITRE ATT&CK framework, which is open to expansion. Consequently, the detection rule base in the first stage and the indicator list in the third stage of CPD require updates whenever new attack techniques or behaviors are identified in real world. However, the process of manually updating the rule base in the first stage and the indicator list in the third stage is relatively quick, taking only a few minutes. Furthermore, it is important to note that, in contrast to signature-based IDS that utilize easily alterable hard-coded strings, CPD is founded on characteristic attack behaviors and predominantly employs immutable indicative strings in its first stage. In essence, CPD relies on the MITRE ATT&CK Matrix and high-level behavioral rules that evolve at a considerably

slower rate. By monitoring the modifications made to the MITRE ATT&CK Matrix, we observe that updates occur approximately every six months. Our detection rule base and indicator list are derived from a prior release dated April 25, 2023. Upon reviewing the persistence tactics and techniques in the latest release (April 23, 2024), we conclude that no updates to CPD are necessary.

**Adaptability and Generality of CPD.** CPD is founded on the MITRE ATT&CK Matrix, a widely recognized framework appreciated by organizations worldwide. Consequently, prominent security vendors not only contribute to this framework but also utilize it as a reference point for developing detection rules and mechanisms. Additionally, we employ well-known standard instrumentation-free logging frameworks to gather system logs. Our CPD prototype integrates with Elasticsearch, a leading tool for event search and threat analysis utilized by many organizations. CPD has been tested within a typical Windows Domain network, which includes both Windows and Linux machines as monitored clients, in accordance with the requirements outlined in the MITRE emulation plans. The deployment of CPD occurs on a Linux server that processes system logs received from client machines and conducts attack detection. These features, along with its strong foundation in the MITRE ATT&CK framework, position CPD as a flexible and comprehensive solution, easily implementable across various enterprise environments for effective detection of persistent threats with minimal implementation effort.

**Completeness of CPD.** The methodology of CPD, as demonstrated through particular examples and MITRE techniques in this chapter, represents a thorough and adaptable framework for identifying persistence threats. The analysis conducted on a case-by-case basis is not merely a collection of isolated examples; rather, it serves as indicative samples of more extensive persistence threat patterns, highlighting CPD's applicability across various threat environments. This strategy guarantees that, although the examples presented may seem specific, the foundational principles – such as the division into setup and execution phases – are broadly relevant. Such an approach emphasizes the comprehensiveness of CPD, confirming its ability to tackle not only established scenarios but also to evolve in response to new threats. Furthermore, CPD's efficacy is substantiated through rigorous evaluation across a range of datasets, demonstrating its superior attack detection capabilities and graph completeness in comparison to state-of-the-art methods.

## 7.6 Summary

In this chapter, we present CPD, a novel system designed specifically for the detection of persistence attacks. Distinctively, CPD utilizes provenance analytics, advancing beyond the conventional detection rules that have been typically employed. This methodology not only greatly minimizes false positives but also significantly improves the accuracy in recognizing authentic persistence attacks. Our assessments, carried out on both publicly available datasets and those generated from meticulously executed MITRE emulation plans, illustrate CPD's superiority compared to state-of-the-art detection methods. Additionally, CPD operates with minimal runtime overhead, rendering it a beneficial enhancement to the array of threat detection tools in enterprise environments. CPD is currently evaluated against persistence techniques on Windows and Linux. We leave it to future work to evaluate CPD against persistence techniques on other OS like MacOS.

# 8 Cross-Machine APT Detection and Investigation

We stress in Chapter 2 that, due to network segmentation, a series of cross-machine activities / lateral movements are necessary for APT actors to reach target devices for gathering and exfiltrating sensitive data and/or impairing physical processes. The most typical way to move laterally inside a victim network is through some form of stolen domain credentials, e.g., password hashes, cached authentication tickets. These cross-machine attack types mostly exploit the complex authentication process in a domain environment via an identity and access management (IAM) system. Microsoft Active Directory is the most widely deployed IAM system across the globe. In this chapter, we present our second APT detection and investigation system HADES, specifically targeting cross-machine APT attacks. HADES is, to the best of our knowledge, the first provenance system achieving accurate and efficient causality-based cross-machine tracing in enterprise networks. Although HADES is designed for detecting and investigating Active Directory-based cross-machine APT attacks, its principles are applicable or adaptable for networks employing a different IAM system.

This chapter is based on a previous paper of ours. Note that, to differentiate our own works from others, we cite our own works with a leading letter "P". Inside the chapter, these kind of citations are used in particular for tables, figures and algorithms taken or adopted from our previous paper:

[P5] Q. Liu, K. Bao, W. U. Hassan, and V. Hagenmeyer, *HADES: Detecting Active Directory attacks via whole network provenance analytics*, 2024. arXiv: 2407.18858 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2407.18858, under review

## 8.1 Challenges & Objectives

Provenance-based intrusion detection systems (PIDS) [22]–[36] have proved to be effective in particular in reducing false alarm rates and presenting real attack activities in attack graphs. Attack graphs provide much richer context for security analysts to further investigate the scope of an attack, invaluable for an effective and efficient security operation center (SOC). However, prior PIDS are constrained to intra-machine provenance tracing, and lack the ability to track across machines in an enterprise environment, and accurately reveal the scale of attackers' traversal inside the network, crucial for attack recovery & remediation.

Cross-machine provenance tracing encounters considerable obstacles due to the well-known issue of dependency explosion [110], [111]. Intra-machine dependency explosion typically occurs in long-running processes, where each input is deemed causally responsible for all subsequent outputs, and vice versa. Cross-machine dependency explosion emerges when cross-machine edges are created solely based on network interactions. Simply linking two intra-machine provenance graphs upon a logon event or any network connection between them [31] would lead to a multitude of erroneous dependencies, as elaborated in Section 8.2.3.

Recognizing the significant role of logon session ID, we introduce a novel concept termed *logon session-based execution partitioning and tracing* within our system HADES. HADES utilizes a two-tiered methodology to effectively detect cross-machine attacks based on Active Directory. The initial tier features a lightweight model for detecting authentication anomalies, which identifies possible Active Directory attacks and relays its findings to the second tier of HADES. This second component is responsible for logon session-based tracing and the triage of attack graphs.

Throughout the development of HADES, we encountered several challenges in achieving precise cross-machine tracing. First, depending on remote access type, each authentication & logon process causes varying number of logon events with distinct logon session ID, which complicates the process of identifying the correct session ID. Second, in certain scenarios, activities within a new logon session may be incorrectly assigned an existing session ID, leading to misleading dependencies. Third, it is often unfeasible to determine the type of remote access solely by analyzing the current logon event. To address these issues, we propose the implementation of a *remote access type*

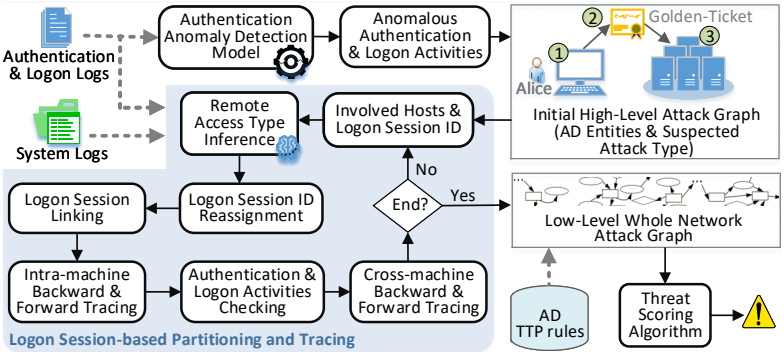



**Figure 8.1:** HADES overview [P5].

*inference* module, a *logon session ID reassignment* module, and a *logon session linking* module within HADES, all developed from our comprehensive profiling and analysis of Windows logging frameworks. Our approach to logon session-based tracing eliminates the need for labor-intensive instrumentation, reduces the risk of errors associated with training, and is intended for the whole system rather than a single program.

## 8.2 HADES Design

Our system HADES employs a two-stage approach for efficient and accurate detection of Active Directory attacks, producing an initial high-level attack graph and a low-level whole network attack graph, respectively. Figure 8.1 provides an overview of HADES, whereas a formal description of HADES's detection procedure is given in Algorithm 4. In its first stage, HADES relies on authentication & logon logs only, and parses through each Active Directory authentication & logon log event. It traces backward on each logon event, and traces backward & forward on each authentication event (Algorithm 4 Lines 29-30), as authentication is a multiple-step process. The core component of HADES's stage one is a lightweight authentication anomaly detection model, against which the authentication & logon tracing result is matched. Once an anomaly is found, HADES produces a high-level attack graph including involved Active Directory entities (Algorithm 4 Line 9), i.e., users and machines, and labeled with a suspected Active Directory attack

type (Algorithm 4 Line 32), at the end of its stage one. A concrete example of such high-level attack graphs is given in Figure 8.2, in which it shows that user Alice from the Exchange Server has used user Bob's password hash to authenticate against the Domain Controller, and then accessed the Data Server, mimicking the Pass-the-Hash behavior.

The identified host names, user names and their logon session ID are passed to HADES's stage two, in which it performs logon session-based execution partitioning and tracing. On the accessed host, e.g., the Data Server in Figure 8.2, by leveraging both authentication & logon logs and system logs, HADES first infers the remote access type, which is critical for deciding whether the logon session ID needs to be reassigned in the next step. Then it links related logon sessions resulted from the same identity, and performs intra-machine backward & forward tracing inside these logon sessions (Algorithm 4 Lines 14-15). Afterwards, HADES checks the authentication & logon logs to spot any logon event inside any domain-joined machine initiated from the accessed host, i.e., cross-machine forward tracing. Meanwhile, HADES examines the authentication & logon logs to find out whether and from which domain-joined machine the current logon session inside the accessing host, e.g., the Exchange Server in Figure 8.2, is initiated, i.e., cross-machine backward tracing. After the logon session-based tracing ends as no further involved machine can be found, HADES passes the low-level whole network provenance graph to its threat scoring algorithm (Algorithm 4 Line 22), the final step of stage two. In this step, we leverage open-source TTP (Tactics, Techniques, Procedures) detection rules [74] for Active Directory discovery and credential access.

**Algorithm 4:** Active Directory Attack Detection [P5]

**Inputs :** System audit log events $\mathcal{E}$;
Authentication & logon events $\mathcal{A}$;
AD TTP rules $\mathcal{R}$

**Output:** List $L_{<AG,TS>}$ of attack graph and its threat score pairs

**Function** GetAdAttackGraph($\mathcal{E}$, $\mathcal{A}$)
  /* Get a list of authentication & logon anomalies */
  $L_{<\mathcal{A}\gamma,\mathcal{L}\gamma,\mathcal{T}\gamma>} \leftarrow$ GetAuthenticationAnomaly ($\mathcal{A}$)
  **foreach** $(\mathcal{A}\gamma, \mathcal{L}\gamma, \mathcal{T}\gamma) \in L_{<\mathcal{A}\gamma,\mathcal{L}\gamma,\mathcal{T}\gamma>}$ **do**
    $AG \leftarrow$ null
    $HG \leftarrow$ CreateHighLevelGraph ($\mathcal{A}\gamma$, $\mathcal{L}\gamma$)
    $AG \leftarrow AG \cup HG$
    $AccessType \leftarrow$ CheckRemoteAccessType ($\mathcal{A}\gamma$, $\mathcal{E}$)
    $SessionID \leftarrow$ ReassignSessionID ($AccessType$, $\mathcal{A}\gamma$, $\mathcal{E}$)
    $LinkedSessionID \leftarrow$ LinkSessions ($SessionID$, $\mathcal{A}\gamma$, $\mathcal{E}$)
    $(\mathcal{E}\alpha, G\alpha) \leftarrow$ TraverseBackward($SessionID$, $LinkedSessionID$, $\mathcal{E}$)
    $(\mathcal{E}\kappa, G\kappa) \leftarrow$ TraverseForward($SessionID$, $LinkedSessionID$, $\mathcal{E}$)
    $AG \leftarrow AG \cup G\alpha \cup G\kappa$
    $\mathcal{E}\alpha \leftarrow \mathcal{E}\alpha \cup \mathcal{E}\kappa$
    $\mathcal{A}\alpha \leftarrow$ CheckAuthenticationLogon($\mathcal{E}\alpha$, $\mathcal{A}$)
    **if** $\mathcal{A}\alpha$ *is not null* **then**
      **goto** 11
    **end**
    $TS \leftarrow$ GetThreatScore ($AG$)
    $L_{<AG,TS>} \leftarrow L_{<AG,TS>} \cup (AG, TS)$
  **end**
  **return** $L_{<AG,TS>}$

**Function** GetAuthenticationAnomaly($\mathcal{A}$)
  **foreach** $ae \in \mathcal{A}$ **do**
    $\mathcal{A}_\alpha \leftarrow$ TraversalBackward($ae$)
    $\mathcal{A}_\kappa \leftarrow$ TraversalForward($ae$)
    $\mathcal{A}_\alpha \leftarrow \mathcal{A}_\alpha \cup \mathcal{A}_\kappa$
    $(\mathcal{L}\alpha, \mathcal{T}\alpha) \leftarrow$ CheckAttackType($\mathcal{A}_\alpha$)
    **if** $\mathcal{L}\alpha \in L_{<\mathcal{L}_{attack}>}$ **then**
      $L_{<\mathcal{A}\gamma,\mathcal{L}\gamma,\mathcal{T}\gamma>} \leftarrow L_{<\mathcal{A}\gamma,\mathcal{L}\gamma,\mathcal{T}\gamma>} \cup (\mathcal{A}\alpha, \mathcal{L}\alpha, \mathcal{T}\alpha)$
    **end**
  **end**
  **return** $L_{<\mathcal{A}\gamma,\mathcal{L}\gamma,\mathcal{T}\gamma>}$

**Function** GetThreatScore($AG$)
  **foreach** $(CrossMachineEdge, \mathcal{T}\omega) \in AG$ **do**
    /* Get a list of discovery techniques and frequency */
    $L_{<tec_d,freq>} \leftarrow$ CheckADDiscovery($AG$, $\mathcal{T}\omega$, $\mathcal{R}$)
    $L_{<tec_{ca},freq>} \leftarrow$ CheckCredentialAccess($AG$, $\mathcal{T}\omega$, $\mathcal{R}$)
    $L_{<pe>} \leftarrow$ CheckPrivilegeEscalation($AG$)
    $TS_{sub} \leftarrow$ CalculateScore($L_{<tec_d,freq>}$, $L_{<tec_{ca},freq>}$, $L_{<pe>}$)
    $L_{<TS_{sub}>} \leftarrow L_{<TS_{sub}>} \cup TS_{sub}$
  **end**
  $TS \leftarrow$ SumScore($L_{<TS_{sub}>}$)
  **return** $TS$

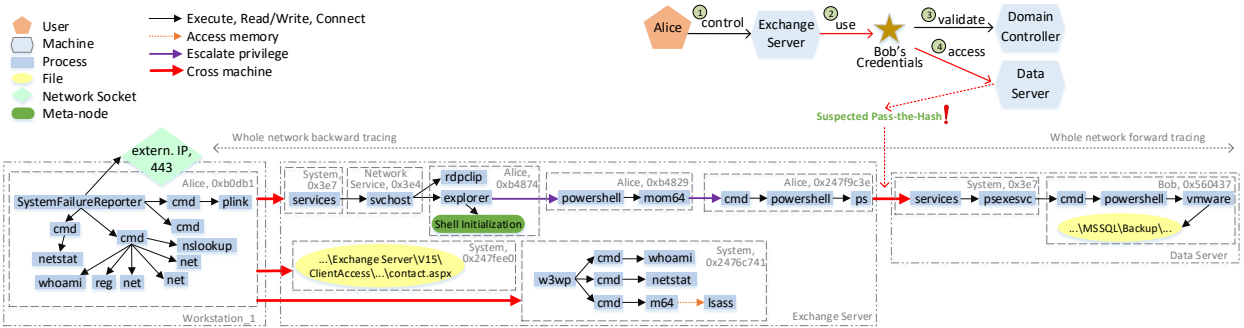



**Figure 8.2:** An Active Directory attack graph created by HADES on the AVIATOR-Oilrig dataset [P5]. HADES first creates an initial high-level attack graph involving Active Directory entities like users and hosts, after it detects an authentication & logon anomaly and suspects a Pass-the-Hash attack. Then it performs system-level forward tracing inside the specific logon session under user Bob in the accessed host Data Server, and system-level forward & backward tracing inside the logon session of Alice in the accessing host Exchange Server. Subsequently, it traces back to a logon session of Alice in the Workstation_1. Next, it traces forward & backward inside this logon session, leading to another two logon sessions in the Exchange Server. This graph reveals that an attacker leveraged a C2 (Command and Control) agent disguised under the process name SystemFailureReporter on the Workstation_1 to perform Active Directory discovery via LOLBins like `net` and `netstat`. The attacker then pivoted to the Exchange Server, performed further Active Directory discovery, and conducted credential access, before moving to the Data Server for accessing critical data.[1]

In the example of Figure 8.2, the whole network forward tracing from the accessed host, i.e., the Data Server, did not lead to further logon session on another machine, as the attacker found the targeted data and did not advance further in the network. However, the whole network backward tracing from the accessing host, i.e., the Exchange Server, reveals that the current logon session on it is a RDP (Remote Desktop Protocol) session that was initiated from another logon session of user Alice inside another machine Workstation_1. Backward tracing from the Workstation_1 did not lead to further domain-joined machine, but rather an external IP address, which belongs to the C2 (Command & Control) server operated by the attacker. Forward tracing from the Workstation_1 resulted in another two logon sessions in the Exchange Server preceding the RDP logon session on it. In one of these logon sessions, credential access was performed, contributing to the later lateral movement from the RDP logon session on the Exchange Server to a logon session on the Data Server. Note that backward tracing can only lead to exactly one logon session, while forwarding tracing can lead to multiple logon sessions, as one may remotely access multiple machines from one machine and multiple logon sessions inside one machine do not interfere with each other.

### 8.2.1 Active Directory Attack Skeleton

A proper understanding of the prerequisites for different Active Directory attacks, along with the associated logging capabilities and constraints, is essential for effective detection of such attacks. Figure 8.3 illustrates the skeleton of Active Directory attacks, detailing the most relevant attack techniques. This model acts as a foundational guide for reliable identification of Active Directory attack techniques and the reconstruction of high-level Active Directory attack graphs. These graphs can subsequently be elaborated to form whole network attack graphs, encompassing all malicious activities executed by attackers within the system.

[1] Note that the bounding boxes with a session ID label or host name are manually added for better readability, this information is encoded in nodes in the original attack graph. Besides, we replaced the user names in the original emulation plan with shorter names. Some processes, e.g., ps and vmware, in the attack graph are malicious processes disguised under a false name. Further, we introduced a graph optimization technique called meta-nodes to conclude system activities of standard known benign routines.

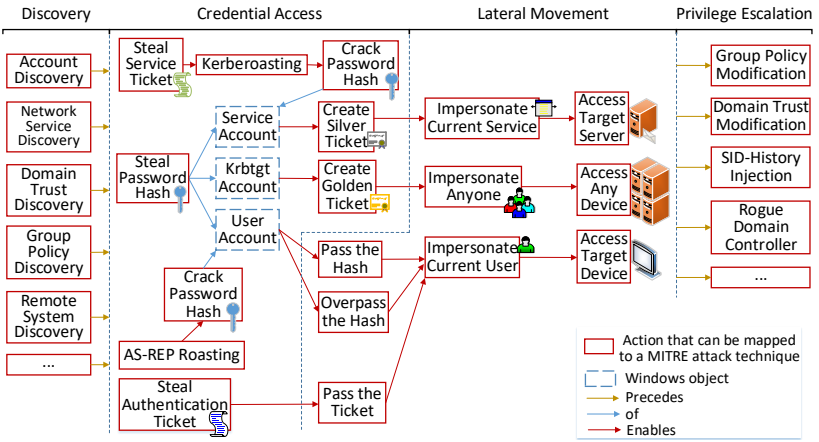



**Figure 8.3:** Active Directory attack overview [P5].

### 8.2.2 Authentication Anomaly Detection

The main purpose of Active Directory is providing the service of authentication of user or machine accounts and authorization to different network resources. Active Directory employs Kerberos as its default authentication protocol, and implements it in two components in a domain controller (DC), i.e., the Authentication Service (AS) and the Ticket-Granting Service (TGS). Figure 8.4 shows a simplified version of standard Active Directory authentication process: 1) the user sends an encrypted AS request for a TGT (Ticket-Granting-Ticket), 2) the DC replies with a TGT if it can decrypt the request and hence verify the user's identity, 3) the user asks for a TGS ticket by providing the TGT, 4) the DC replies with a TGS ticket, 5) the user asks for access to an application server by providing the TGS ticket, 6) the server gives the user requested access after validating the TGS ticket. The entire process uses shared secret cryptography to deter network-level eavesdropping and replay attacks, meaning that credential access is only possible inside hosts.

The complex process of network authentication is frequently targeted by attackers who succeed in acquiring some form of credentials or credentials-related data from one machine, referred to as credential access. These stolen credentials are then utilized to authenticate against a domain controller, enabling the attackers to gain access to additional machines, a tactic known as

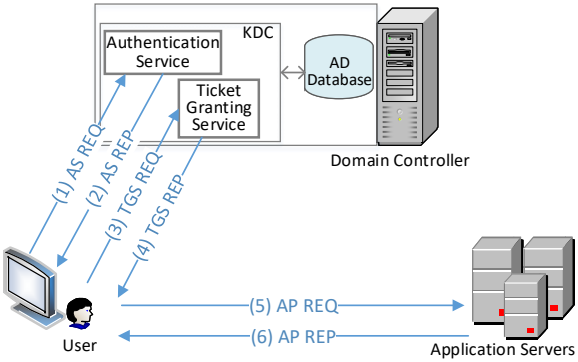



**Figure 8.4:** Standard Active Directory authentication process [P5].

lateral movement. Various techniques employed in Active Directory attacks share common characteristics, which can lead to confusion. Nevertheless, it is our assertion that an effective detection system must distinguish between these different types of attacks to appropriately prioritize alerts, given the varying levels of severity associated with each. For example, a Golden Ticket [202] attack is considered more severe than other attack types, as it has the potential to result in a complete compromise of the domain.

We analyzed each attack type, and identified anomaly in the corresponding authentication process, as depicted in Figure 8.5. For example, in AS-REP Roasting attacks, the attackers issue a TGT request with the intention of offline cracking the user's password from the obtained TGT, resulting in the absence of a subsequent TGS request. In the case of Kerberosting attacks, the objective is to decipher a service account's password from a received TGS, which indicates that there is no access to the application server following the TGS request. Regarding Pass-the-Ticket attacks, the attackers submit a TGS request to a Domain Controller using a stolen TGT, signifying that no prior TGT request occurs before the TGS request.

We propose a lightweight model for detecting authentication anomalies based on our findings. As demonstrated in our evaluation in Section 8.4, this model effectively identifies each instance of attack; however, it suffers from a significant false positive rate, which limits its practical applicability. The causes of these false positives are varied. For example, network disruptions can lead to false positives in AS-REP Roasting and Kerberosting. Additionally,

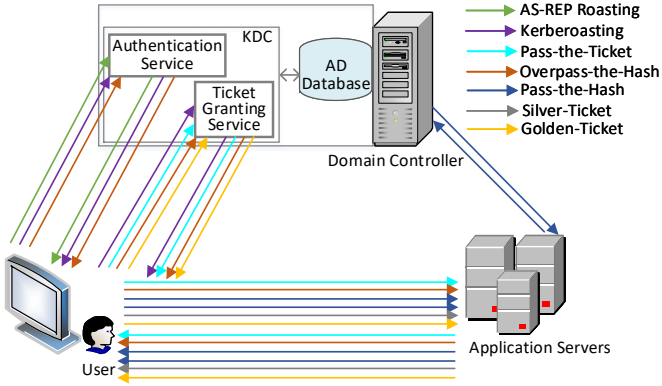



**Figure 8.5:** Authentication incompleteness/abnormality [P5].

the legitimate use of native Windows applications such as `runas` can trigger erroneous alerts for Pass-the-Hash attacks.

### 8.2.3 Logon Session-based Execution Partitioning and Tracing

Recent PIDS have proved to be invaluable in reducing false alarms. To tackle the high rate of false positives, we aim to develop the first causality-based cross-machine PIDS. A naive approach would be to connect the intra-machine provenance graphs of two domain-joined machines with a cross-machine edge, whenever there is a network connection between them, as suggested in [31]. However, this kind of cross-machine edges do not represent a causality, but rather a correlation, leading to numerous cross-machine edges between intra-machine provenance graphs. Take Figure 8.2 as an example, this would result in 7364 edges between the Exchange Server's provenance graph and other hosts' graphs after only one day operation, while in fact there are only 4 causality-based cross-machine edges to/from the Exchange Server in the true attack chain.

A more advanced approach would be to connect intra-machine provenance graphs, whenever there is a logon event from one machine to another. Yet, due to the prevalence of credential thefts, in particular in Active Directory attacks, it is impossible to identify the true identity behind each logon. For instance, in Figure 8.2, the attacker has stolen Alice's credentials. Given that both Alice

and the attacker have accessed the Exchange Sever from the Workstation_1 using the same credentials, this approach would still create false-dependencies. In fact, this approach would produce 479 correlation-based cross-machine edges, instead of 4 causality-based, to/from the Exchange Server. Correlation-inferred edges are the root cause of the notorious dependency explosion problem [109]–[111].

In order to enable causality-based provenance tracing, we resort to a critical yet previously undiscovered information: logon session ID, and propose a novel concept: *logon session-based execution partitioning and tracing*. To the best of our knowledge, this work presents the first PIDS leveraging logon session ID. Windows assigns a logon session ID (unique until next reboot) after each successful authentication & logon to label system activities run in that session in Windows Security logs [155]. A logon session is a computing session assigned with an access token representing the authenticated account's security context and permissions [203]. Every Windows process runs within a logon session's context.

**Benefits.** The benefits of logon session-based tracing are fourfold: 1) it enables fine-grained causality-based cross-machine provenance tracing, 2) it alleviates dependency explosion also for intra-machine provenance tracing, 3) it reduces log size as most forensics-irrelevant system activities under logon sessions with predefined logon ID[2] can be safely discarded, 4) it automatically pinpoints privilege escalation during intra-machine provenance tracing.

When a user from one machine logs into another machine[3], the corresponding logon session in the second machine is resulted from exactly one logon session in the first machine. For example, in Figure 8.2, Alice's logon session `0xb4874` on the Exchange Server is only resulted from Alice's logon session `0xb0db1` on the Workstation_1, while there are multiple Alice's logon sessions on the Workstation_1 in parallel. Our system HADES can accurately disclose this, as it checks authentication & logon logs among involved machines and performs logon session-based tracing. *With logon session-based tracing, we can create a whole network provenance graph representing activities conducted by exactly one true identity, be it the attacker or a genuine user.*

[2] Predefined logon ID include `0x3e4`, `0x3e5`, and `0x3e7`, belonging to Network Service account, Local Service account, System account, respectively [204].

[3] Note that a user accessing resources on a remote machine also triggers a logon on the remote machine. The authentication & logon process is transparent to users due to Single Sign-on.

The benefit is even more apparent during intra-machine provenance tracing. An enterprise-level application server typically hosts more than a hundred logon sessions at the same time, with some belonging to the same user, and runs cumulatively thousands of logon sessions after just one day operation. For instance, in the AVIATOR-Oilrig [89] dataset, system activities run on the Exchange Server spread over more than 5 thousands logon sessions. While some logon sessions are (interactive) long-running ones, many others die soon after a short execution, e.g., accessing some resources. Oblivious to logon session ID, prior provenance tracing approaches would connect all these system activities, as they all at the end can be traced back to the root process, `system` (PID=4) on Windows. In contrast, our approach identifies the truly involved logon sessions and tracks system activities only inside them, greatly easing the dependency explosion problem.

The efficiency introduced by leveraging logon session ID shows not only during provenance tracing, but also for log storage reduction. Under the assumption of OS integrity, most system activities run under logon sessions with predefined logon ID belong to standard routines, and can be considered as forensics-irrelevant and safely discarded. In the example of the Exchange Server in the AVIATOR-Oilrig dataset, these system activities account for over 90% of all log events created on it. However, some of these system activities are forensics-relevant, as they are related to remote access and logons. HADES identifies and tracks these system activities during its logon session linking. Further, as logon sessions also introduce the separation of privileges, HADES automatically identifies privilege escalation during intra-machine cross-session tracing by checking the value of "Token Elevation Type" and "Integrity Level" associated with a logon session ID. By doing so, it has identified, e.g., in Figure 8.2, the privilege escalation from unprivileged user to Administrator, i.e., from `0xb4874` to `0xb4829`, and then privilege escalation from Administrator to System, i.e., from `0xb4829` to `0xb247f9c3e`.

**Challenges.** We encountered several challenges while developing HADES. First, each network authentication & logon process results in multiple logon events with distinct logon session ID on the accessed machine. Depending on the remote access type, e.g., via RDP or SMB (Server Message Block), the number of logon sessions created varies. Second, under certain circumstances, system activities resulted from a new logon are recorded with an existing session ID, leading to false dependency. Third, it is often not possible to reveal the remote access type by inspecting the logon event alone.

**Remote access type inference.** Identifying remote access type is critical for logon session ID reassignment and session linking, both crucial for accurate cross-machine tracing. The way how exactly Windows emits logon events and assigns logon ID is obscure. Due to Windows' closed-source nature, revealing this via source code is not possible. Reference to the most detailed sources about Windows [182], [204]–[206] also failed to deliver an answer. Hence we resorted to extensive testing and analysis of each remote access type.

We found that both authentication & logon events and system activities events need to be processed to extract a list of attributes for accurately inferring remote access type. Hence, for each logon event, HADES checks related authentication & logon events and system events to analyze the context and then classify the remote access type. HADES can identify all standard remote access types: RDP, SSH, Powershell-Remoting (WinRM), WMI, RPC, PsExec, Network Share (SMB), internal web-server request. Note that only identity-based remote access types are of interest, in which valid accounts are used to access a computer. Remote access via malware/C2 agent is not identity-based, and will not create new logon sessions. Repetitive accesses via the same malware/C2 agent run in the same logon session and their system activities are recorded with the same session ID. That is, as long as HADES can trace to the session in which the C2 agent is present, it can detect all system activities conducted by it, without the need for inferring remote access type and session linking etc. For example, in Figure 8.2, the attacker has performed several attack steps via the C2 agent on the Workstation_1 at different times, and all of these steps are detected by HADES, as they are all executed in the same session `0xb0db1` and HADES has successfully traced back to this session.

We also found that some logon events and associated logon sessions are purely byproducts. Some of these sessions terminate soon after being created and others live until a logoff occurs. These logon sessions typically do not contain any forensics-relevant system activities. Although their creations are resulted from a user logon, they are not influenced by the logged-on user. For instance, when a domain user logs into a machine via SSH, a byproduct logon session under a virtual user *sshd_xxx* gets created.

**Logon session ID reassignment.** For several remote access types, system activities performed on a new logon session are recorded under an existing logon session ID, necessitating session ID reassignment for causally correct tracing. A typical example is RDP, which is often misused by attackers [207].

Unlike SSH, a remote logon session via RDP in the accessed machine will not terminate, when the client program on the accessing machine exits. The user has to explicitly log out to terminate that session. However, users typically close the client program, unaware of their still-running logon sessions on the remote machine.

When the user's credentials are stolen by an attacker who then logs into the same remote machine with the same credentials, all system activities conducted by the attacker are labeled with the ID of the user's long-running session, although a new logon session with a different ID is created. This is due to Fast User Switching [208], which provides users the same setup after reconnecting to an existing local console session or remote desktop session[4]. The new logon session starts with the new logon by the attacker, and ends when the attacker disconnects (not necessarily explicitly logs off), marking the exact timespan of system activities conducted only by the attacker. Note that it is not possible for the attacker and the user to be on the same remote desktop session (and hence the same logon session) at the same time, a restriction enforced on Windows OS. Hence, if HADES identifies a logon as RDP access, it further checks for another specific related event indicating whether the user has entered a new remote desktop session or an existing one. If an existing remote desktop session (and hence the logon session) is being reentered, HADES takes the ID of the new logon session and replaces the existing logon session's ID (inside the remote desktop session) with it for the corresponding system activities.

Other remote access types requiring session ID reassignment include Powershell-Remoting and internal web-server request. In Figure 8.2, the attacker from the Workstation_1 leveraged web-shell to execute commands on the Exchange Server. However system activities caused by the attacker's web requests are recorded with System account's logon session ID, i.e., `0x3e7`, along with system activities resulted from other domain users' web requests. HADES correctly identified the remote access type and reassigned the attacker's system activities with `0x2476c741`, separating them from unrelated system activities caused by other users.

[4] Note that logon sessions and local console / remote desktop sessions are two different concepts. Only a few logon sessions "live" in local console / remote desktop sessions, which could be seen as a container for one or a few logon sessions.

**Logon session linking.** Although most system activities under predefined logon ID are forensics-irrelevant, some are directly responsible for the creation of users' logon sessions. For instance, when a user from one machine executes remote commands on another machine via Powershell-Remoting, the remote commands are executed by the process `wsmprovhost.exe` on the second machine, which is spawned by an instance of `svchost.exe` process under the System logon session, i.e., `0x3e7`, once the user successfully authenticated against a domain controller. HADES links users' logon sessions with those predefined logon sessions, and performs only backward-tracing in those sessions until finding the root process responsible for the creation of the user logon session of each access type. Logon session linking is also needed when a privileged user logs in a remote machine via RDP. Due to UAC [209], two logon sessions are created inside a remote desktop session, of which one has a privileged access token and the other does not. System activities run under both logon sessions can only result from the same true identity. Hence these two sessions need to be linked for accurate tracing.

### 8.2.4 Threat Score Assignment

With our summarization of Active Directory attacks in Figure 8.3, we identified two key insights for accurate Active Directory attack detection. The first insight is that attacks against Active Directory follow a rigid pattern: Active Directory discovery, credential access, lateral movement and privilege escalation. The second insight is that the key enabler of an Active Directory attack is successful credential access, and the most observable result of an Active Directory attack is lateral movement. For instance, in Kerberoastig attacks, a service must be first identified, whose credentials will be then stolen, and used for lateral movement and privilege escalation afterwards.

Based on these two critical insights, HADES calculates a threat score for each provenance attack graph generated after logon session-based backward & forward tracing. This is to ensure that the most likely true attacks are always investigated first by security analysts. For each potential lateral movement, i.e., a cross-machine edge in an attack graph, HADES checks how many credential access techniques were executed in the corresponding hosts before, and how often each technique was performed. For each performed credential access technique, HADES explicitly checks whether the corresponding process has accessed the system process `lsass.exe`, which is the most critical process

for managing credentials like password hashes and access tokens on Windows OS, and is hence often abused by attackers. Credential access techniques involving accessing `lsass.exe`'s memory should be considered more severe. Then, HADES examines how many Active Directory discovery techniques were operated on related hosts and the frequency of each discovery step conducted. Also, HADES inspects whether and how many times privilege escalation is observed in the provenance attack graph.

We formulate the threat score calculation in the following two equations.

$$TS_E = \sum_{i=1}^{n} (Freq(teq_i) \times Var(tac_i))^{w_i} \tag{8.1}$$

where $n$ denotes the number of tactics involved, e.g., credential access. $Freq(teq_i)$ denotes the accumulated execution frequency of techniques in a given tactic, and $Var(tac_i)$ denotes the number of techniques being applied for the given tactic. Considering both the intensiveness $Freq(teq_i)$ and the extensiveness $Var(tac_i)$ of a tactic being performed aligns with the fact broadly observed in threat reports [210] that attackers often apply multiple techniques within the same tactic to boost their chances of success. Besides, $w_i$ is a weighting factor introduced for each tactic. Credential access is considered as the most relevant tactic, and receives a higher weight than discovery and privilege escalation, as per our second insight.

Whereas Equation 8.1 calculates the threat score for each cross-machine edge, Equation 8.2 accumulates the threat scores returned from Equation 8.1, and outputs the threat score for the attack graph.

$$TS_G = \sum_{i=1}^{n} (TS_i \times (DA + 1) \times Criticality) \tag{8.2}$$

where $n$ denotes the number of cross-machine edges found in a given attack graph, and $TS_i$ denotes the threat score assigned for each cross-machine edge from Equation 8.1. $DA$ indicates whether credentials of domain administrators are involved in the cross-machine system activities, and takes the value 0 or 1. By doing so, we prioritize attack graphs for further investigation, in which credentials of domain administrators are involved due to more severe consequence of these attacks. Similarly, we assign a $Criticality$ to different

attack types. Because, for instance, a Golden-Ticket attack may imply a compromise of the entire domain, whereas a Silver-Ticket attack's scope is limited to certain servers in the domain. If credentials of a non-privileged domain user are involved in a Pass-the-Hash attack, the consequence is even less critical.

## 8.3 Implementation

We implemented a prototype of HADES in Python, and deployed it on a 64-bit Ubuntu 22.04 OS with 256 GB of RAM and a 32-core processor. This machine also hosts Elasticsearch [169], to which HADES interfaces via EQL [175], a query language designed for security purposes. Note that, for enhanced efficiency, HADES functions as an on-demand tracing system, which searches for relevant events to process and outputs attack graphs only after an authentication anomaly is detected in its first stage. Elasticsearch provides scalable and near real-time search for log data investigation. In Active Directory Domain, authentication logs are recorded only on domain controllers, whereas logon logs are collected in the accessed hosts. We collect authentication logs from the domain controller and logon logs from each domain-joined host. Authentication logs and logon logs can be linked with a logon GUID. For system activities, we collect Windows Security [155] logs and Sysmon [84] logs; both are industry-standard logs.

## 8.4 Evaluation

**Public Datasets.** Public datasets often do not contain Active Directory attacks presumably due to higher requirement on setting up emulation infrastructures. For instance, the DARPA-E3 dataset [69] does not contain any Active Directory-based attack. Although in the DARPA-E5 dataset [70], a so-called Copykatz module is used for credential access, the credentials obtained are local credentials, as the corresponding machines are not joined to a domain. The DARPA-OpTC dataset [71] is the only public dataset including Active Directory-based attacks that we can find. Unfortunately, this dataset only includes system logs on domain-joined hosts, but no logs from the domain controller. Authentication logs, which are recorded only in domain

**Table 8.1:** Overview of HADES's evaluation datasets [P5].

| Dataset | Target Host Number | Ground Truth Attack | Target Host OS | Data Size | Event Number |
|---|---|---|---|---|---|
| **AVIATOR-APT29** | 3 | Golden-Ticket | Windows | 253GB | 160M |
| **AVIATOR-WizardSpider** | 3 | Kerberoasting | Windows | 151GB | 87M |
| **AVIATOR-Oilrig** | 4 | Pass-the-Hash | Windows | 200GB | 116M |

controllers, are critical for HADES's functionality. Hence we cannot evaluate HADES on the DARPA-OpTC dataset either.

**MITRE Attack Emulation.** Active Directory attacks are present in nine out of MITRE's eleven full emulation plans [46]. MITRE Engenuity ATT&CK® Evaluations [147] use these plans to assess commercial detection systems from leading security vendors. MITRE's emulation plans target enterprise networks with more sophisticated, cross-machine attacks than most public datasets, offering greater authenticity. However, MITRE has not published any corresponding datasets. We stringently implemented three emulation plans, i.e, APT29 [86], Oilrig [89], and WizardSpider [88], and then evaluated HADES on the created datasets, named AVIATOR-APT29, AVIATOR-Oilrig, and AVIATOR-WizardSpider, respectively. These datasets are included in the AVIATOR dataset as subsets. Table 8.1 gives an overview of our datasets.

### 8.4.1 HADES vs. SIEM Detection Rules

We extracted all detection rules for Active Directory-based attacks from the most popular and established SIEM rule repositories, i.e., Elastic [146], Sigma [74], Google Chronicle [75], and converted them into EQL queries, which we then run parallel to HADES on our datasets. We compare the detection results of Elastic, Chronicle, Sigma and our system HADES in Table 8.2. To ensure a fair comparison, in Table 8.2, we only show the detection results of these SIEM rules for attack techniques that HADES's stage one can detect (Figure 8.5), and not other AActive Directory attack techniques like Active Directory discovery, which would lead to a much higher number of false positives (FP).

**Table 8.2:** Comparison of HADES and prior detection systems [P5].

| Dataset[a] | HADES | | | | Elastic | | Chronicle | | Sigma | | CAD | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Stage 1[b] | | Stage 2 | | | | | | | | | |
| | FP | FN | FP | FN | FP | FN | FP | FN | FP | FN | FP | FN |
| AVIATOR-APT29 | 60 | 0 | 0 | 0 | 148 | 1 | 14 | 1 | 545 | 0 | 0 | 0 |
| AVIATOR-WizardSpider | 66 | 0 | 2 | 0 | 194 | 1 | 30 | 1 | 605 | 0 | 0 | 1 |
| AVIATOR-Oilrig | 156 | 0 | 2 | 0 | 213 | 0 | 37 | 1 | 417 | 0 | 0 | 1 |

[a] Note that each dataset has only one true positive (TP=1).
[b] HADES's stage 1 result refers to the detection result after applying only the lightweight anomaly authentication model, while HADES's stage 2 result refers to the detection result after also performing whole network tracing and threat score calculation.

Note that we implement HADES as an on-demand PIDS for enhanced efficiency. Only when HADES's stage one detects a potential lateral movement, it runs its stage two for a whole network investigation, which unfolds other tactics like Active Directory discovery and credential access when they are causally related to the lateral movement. That is, HADES cannot detect an early-stage Active Directory attack, in which the attacker has only performed Active Directory discovery and credential access, but not yet moved to other machines. In particular, for attacks like Golden-Ticket, HADES's stage one cannot detect the creation of a Golden-Ticket, i.e., credential access, but its usage for lateral movement. HADES's stage two checks credential access techniques related to Golden-Ticket, if the stage one detects a Golden-Ticket-based lateral movement. HADES avoids a high rate of false positives by neglecting potential but yet less critical early-stage attack steps.

Table 8.2 reveals that, comparing to HADES, SIEM rules produced more false positives and missed some true attacks. Whereas Chronicle has a relatively low false positive rate in all datasets, it missed the true attack in each dataset. In contrast, Sigma detected all true attacks, at the cost of having a much higher false positive rate. Elastic detected the Pass-the-Hash attack in the Oilrig dataset with less false positives, but missed the attacks in AVIATOR-APT29 dataset and AVIATOR-WizardSpider dataset.

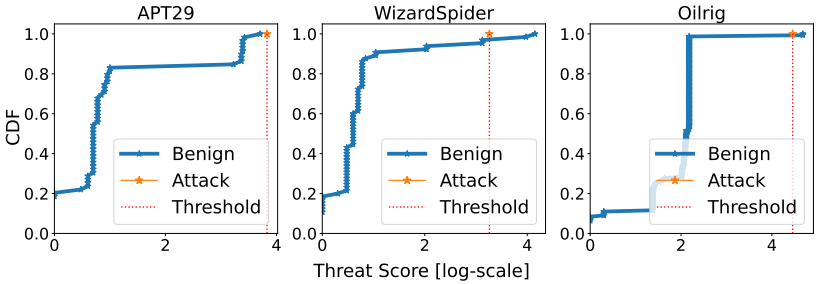



**Figure 8.6:** CDF of threat score for false and true alerts [P5].

To understand why SIEM rules have a high rate of false positives and false negatives (FN), we resort to manually inspect them. On the one hand, we find that a high number of false positives often results from overly general rules matching system activities caused by benign programs often misused by attackers, i.e., LOLBins. Further, Sigma's unusually high number of false positives is also due to the fact that Sigma's rule repository includes many similar rules created by different contributors for the same attack techniques. On the other hand, both Chronicle's and Elastic's rules tend to be overly specific, i.e., having hard-coded strings as conditions, and also often allowlist system activities caused by known "benign" programs. Simply favoring high-degree of specificity and overly allowlisting lead to less false positives, but inevitably cause more false negatives.

Unlike open-source SIEM rules, HADES employs an advanced detection strategy that goes beyond solely looking for specific isolated log events, contributing to an average false positive reduction rate of 98%. Most activities involved in Active Directory attacks are based on (mis)using legitimate Active Directory infrastructures and services. For example, network shares are used intensively by normal domain users and only occasionally by attackers for lateral movement. That is, attackers can easily blend into the noise caused by legitimate users, as indicative system and network activities related to Active Directory attacks are so prevalent in benign operations. However, when putting truly causally related system activities in a provenance graph via precise cross-machine tracing, it exposes provenance graphs containing system activities conducted by attackers as they typically follow a rigid Active Directory attack pattern unlike benign provenance graphs.

Detailed comparison between HADES's stages one and two is given in Figure 8.6, which presents the cumulative distribution function of threat scores for benign and attack alerts. HADES's stage two results in Table 8.2 are based on the true attack threat score threshold. Our threat triage algorithm in HADES's stage two, combined with accurate cross-machine tracing based on logon session-based execution partitioning, contributes to an average false positive reduction rate of 99% when comparing to its stage one.

### 8.4.2 HADES vs. CAD

Prior research [211] shows that, contrary to general perception, commercial security products may have poor detection quality despite high price tag. In order to answer the question how much value a commercial attack detection product could bring, we subscribed to a cloud-based dedicated Active Directory attack detection solution from a popular security vendor and evaluated it alongside HADES on our datasets. The security vendor is considered as a significant security solution provider in the security market overviews of Gartner [212] and Forrester [213]. However we did not get the permission to name the vendor in this work, and hence refer to the security product we purchased simply as CAD (commercial attack detector).

We installed CAD on the domain controller of our emulation infrastructure. We were also provided with a documentation of the security product we purchased, in which a guideline for CAD's logging configuration and its high-level detection logic/rules are included. To our surprise, as shown in Table 8.2, the commercial product did not raise a single false alarm, but is plagued with a high false negative rate. The result of our investigation, however, aligns with the findings in prior work [211], i.e., commercial security products may sacrifice detection rate in exchange for analyst time.

By consulting on the documentation, in particular CAD's high-level detection logic, we were able to find some explanation for the poor detection rate. First, we find that CAD, as a dedicated Active Directory attack detection system, does not collect enough data we consider as necessary for accurate detection. That is, CAD only collects a limited set of authentication & logon event types, leading to restricted visibility. Second, some of its detection rules do not fire alerts on events that are previously observed on the same computers. With the prevalence of LOLBins, doing so reduces false positives, but inevitably introduces false negatives. Third, CAD does not check system logs inside

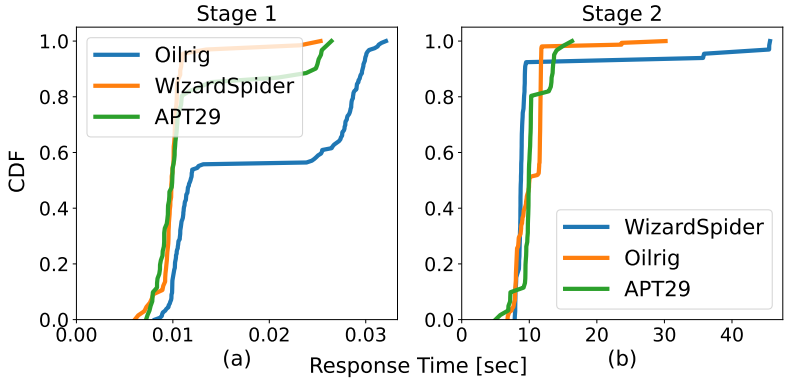



**Figure 8.7:** CDF of response time of HADES [P5].

each host at all. We argue that credential access techniques, a critical step in Active Directory attacks, are detectable mostly via system logs.

To avoid false conclusion, we engaged with the vendor over weeks by first checking if our installation and configuration of CAD was correct, and then presenting our finding, and asking for explanation on the high false negative rate. The answer of the vendor confirmed our evaluation findings. In particular, the vendor responded to the false negative for the Pass-the-Hash attack by saying that CAD's detection on Pass-the-Hash attacks has known issues, and acknowledged ongoing efforts to address these issues.

### 8.4.3 Response Time

We measure the response time of HADES in two separate parts. Figure 8.7 shows the cumulative distribution function of response time of HADES in its two stages, respectively. HADES's stage one response time is measured per authentication & logon alert. Figure 8.7 (a) reveals that it takes less than 35 milliseconds to find an authentication & logon anomaly for all authentication & logon events in each dataset. We measure HADES's stage two response time as the time to perform logon session-based backward & forward tracing on a high-level alert found in the stage one, while checking for system activities related to Active Directory discovery techniques, credential access, and privilege escalation, and calculate the threat score for the resulted attack graph. As shown in Figure 8.7 (b), it takes at most 45 seconds to return a low-level

provenance attack graph with associated threat score for each stage one's high-level alert in every dataset.

### 8.4.4 Case Study

**Pass-the-Hash.** Pass-the-Hash attacks leverage NTLM authentication process, instead of the default Active Directory authentication process Kerberos. This authentication anomaly manifests in our authentication anomaly detection model introduced in Section 8.2. However, using this model alone introduces many false alerts, as NTLM authentication process is still widely in use in Windows domains. For instance, when a user accesses a remote network share via NTLM, the authentication process against the target server and the domain controller looks the same as in Pass-the-Hash attacks. That is, without resorting to system logs, it is difficult to reliably detect Pass-the-Hash attacks. This is also why CAD missed this attack in the Aviator-Oilrig dataset. However, we show that, by using whole network provenance tracing and an alert triage algorithm, Hades can reliably detect Pass-the-Hash, and drastically reduce false positives caused by benign user activities. Figure 8.2 automatically created by Hades presents the entire attack chain conducted by the attacker.

**Golden-Ticket.** In a Golden-Ticket attack, the attacker manages to steal the credentials of the `krbtgt` account from a domain controller, and is then able to create a TGT impersonating any domain user. The attacker does not request a TGT from a domain controller before requesting a TGS to an application server, showing an anomaly in our authentication anomaly detection model. However, during a benign operation, when a cached TGT is used to request access to a server, it exhibits the same network authentication anomaly, making Golden-Ticket difficult to detect without inspecting system logs on each involved machines. For the Golden-Ticket attack in the Aviator-APT29 dataset, as illustrated in Figure 8.8, Hades first creates an initial high-level attack graph involving Active Directory entities like users and hosts, after it detects an authentication & logon anomaly and suspects a Golden-Ticket attack. Then it performs system-level forward tracing inside the specific logon session of involved username Bob in the accessed host Workstation_2, and system-level forward & backward tracing inside the logon session of Alice in the accessing host Workstation_1, leading to a logon session in the Domain Controller. This attack graph discloses that the attacker first

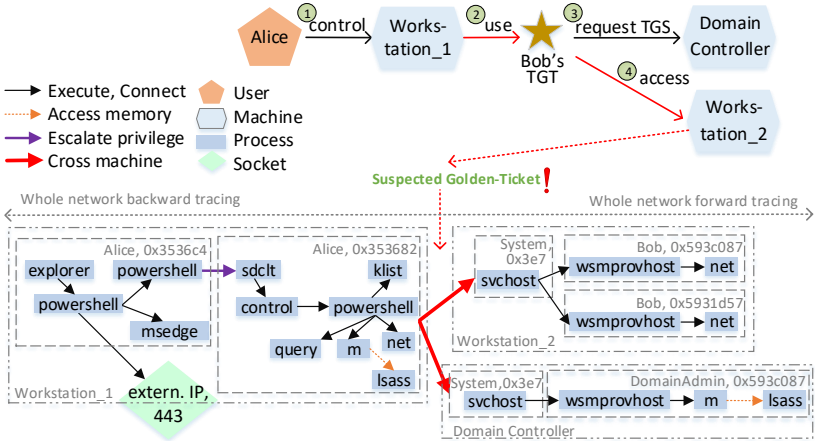



**Figure 8.8:** An attack graph created by HADES on the AVIATOR-APT29 dataset [P5].

performed Active Directory discovery and credential access on the initially compromised machine Workstation_1, then moved to the Domain Controller for further credential access, and finally created a Golden-Ticket for accessing the Workstation_2.

## 8.5 Summary

We present a novel accurate Active Directory attack detection system named HADES in this chapter. Based on a thorough study of Active Directory attacks launched by APT actors, we create a succinct and contextualized Active Directory attack overview revealing key steps and prerequisites of various Active Directory attack types. We leverage critical insights from our extensive analysis on Active Directory attacks to design a lightweight authentication anomaly detection model, and a threat triage algorithm. We propose the concept logon session-based execution partitioning and tracing, which significantly reduces false dependencies, contributing to accurate and swift Active Directory attack detection. Our evaluation, conducted on datasets derived from rigorously implemented MITRE emulation plans, demonstrates that HADES significantly outperforms open-source SIEM detection rules and a commercial dedicated Active Directory attack detector. Yet, AVIATOR's logon session-based tracing

is currently implemented and evaluated for enterprise environments employing Active Directory as the identity and access management (IAM) system. We leave it to future work to assess Aviator's applicability for enterprise environments employing a different IAM system.

# 9 Cross-Machine Multi-Phase APT Detection and Investigation

While Chapter 7 and Chapter 8 present detection and investigation systems for multi-phase APT attacks and cross-machine APT attacks, respectively, this chapter addresses cross-machine multi-phase APT attacks with our third system Commander. The modular design of Commander enables integration of Cpd and another two specialized detectors with Hades. These specialized detectors make complementary contributions to ensuring robust and correct whole network tracing, by delivering critical information to guide and adjust the tracing process. Commander can perform robust tracing for cross-machine multi-phase APT attacks across an industrial-sector organization, for which we additionally develop parsers for logs of popular industrial controllers, and detection rules with a reference to the MITRE ATT&CK Matrix for ICS.

## 9.1 Challenges & Objectives

When targeting industrial-sector organizations, the main objective of APT actors is often impairing physical processes at a *strategic* point. Once achieved initial access on a machine in a target organization, stolen credentials are the most popular way for attackers to access further machines in the network, as enterprise security vendor CrowdStrike's recent studies [37]–[40] show

This chapter is based on a previous paper of ours. Note that, to differentiate our own works from others, we cite our own works with a leading letter "P". Inside the chapter, these kind of citations are used in particular for tables, figures and algorithms taken or adopted from our previous paper:

[P6] Q. Liu, K. Bao, and V. Hagenmeyer, "COMMANDER: A robust cross-machine multi-phase Advanced Persistent Threat detector", under review

that 80% of modern cyberattacks are identity-driven, and OT (Operational Technology) security vendor Dragos's finding [41] suggests that nearly all attackers leverage stolen credentials for lateral movement.

Provenance-based systems parse detailed system logs into provenance graphs, so that TTP security alerts can be linked, contextualized, and then prioritized for further investigation. To evade these systems, APT actors resort to persistence techniques, which enable them to disassemble an attack chain into multiple phases and stay *slow*. This makes it infeasible for provenance-based systems without persistence awareness [22]–[26], [28], [29], [33], [35], [36], [81] to causally link those system activities (and the TTP alerts) from the same attacker, as they are located in fragmented, seemingly unrelated graphs. In Chapter 7, we present CPD, the first specialized persistence detector. CPD can accurately identify attackers' use of persistence techniques from system logs, and hence recouple those disassembled, but actually related graphs.

As discussed in Chapter 4, prior provenance-based IDS (PIDS) are restricted to intra-machine provenance tracing, even though cross-machine activities are omnipresent in real-world APT attacks. This, again, limits PIDS' ability to link attack steps of the same APT actor, and thus reveal the scale of the attacker's traversal inside an organization, which is crucial for accurate attack detection and remediation. To overcome this, we propose a concept called *logon session-based execution partitioning and tracing*, which enables accurate and efficient causality-based cross-machine tracing in enterprise networks, and implement it in HADES.

However, our analysis of APT threat reports linked in the MITRE ATT&CK knowledge base [48] yields the finding that HADES is susceptible to several evasive attack tactics/techniques routinely employed by APT actors: persistence [142], session hijacking [143], and port forwarding [144]. Whereas port forwarding attacks can directly interrupt HADES's cross-machine tracing, both persistence and session hijacking attacks hinder accurate intra-machine tracing, on which cross-machine tracing hinges. The impact of these evasion techniques on HADES is discussed in Section 4.4 in detail.

Recognizing the weaknesses of HADES, we propose COMMANDER, which should integrate CPD with HADES, and incorporate specialized detectors for session hijacking and port forwarding, respectively. The main objective of COMMANDER is to achieve robust, efficient and accurate attack detection and graph reconstruction. Central to this mission is a robust whole network tracing module, which should separate system activities from different identities,

and causally link system activities of the same identity, be it the attacker or a genuine user, on each involved machine across the entire organization. Correct and complete tracing is vitally important for the attack graph ranking, in which graphs are ranked by threat score, determined by the set of indicative attack steps / security alerts found in the corresponding graph. Thus, incorrect or incomplete tracing could let true attack graphs be ranked lower than false-positive ones.

Further, with COMMANDER, we aim to detect cross-machine attacks in industrial-sector organizations consisting of an enterprise network and an industrial control system (ICS), which are typically in two separate domains. That is, COMMANDER should be able to trace cross domains, given that the enterprise network and the ICS network are often connected via a gateway server. Besides, COMMANDER needs to incorporate parsers for system logs from industrial controllers, and TTP detection rules with a reference to the MITRE ATT&CK Matrix for ICS [51] for these controllers. For the evaluation purpose, we choose two popular industrial controllers used in energy systems, i.e., Siemens SIPROTEC [49] and Hitachi Energy RTU500 [50]. COMMANDER's goal is to accurately attribute the activities on these controllers to the true identity behind those actions, even if the access originates from the enterprise network.

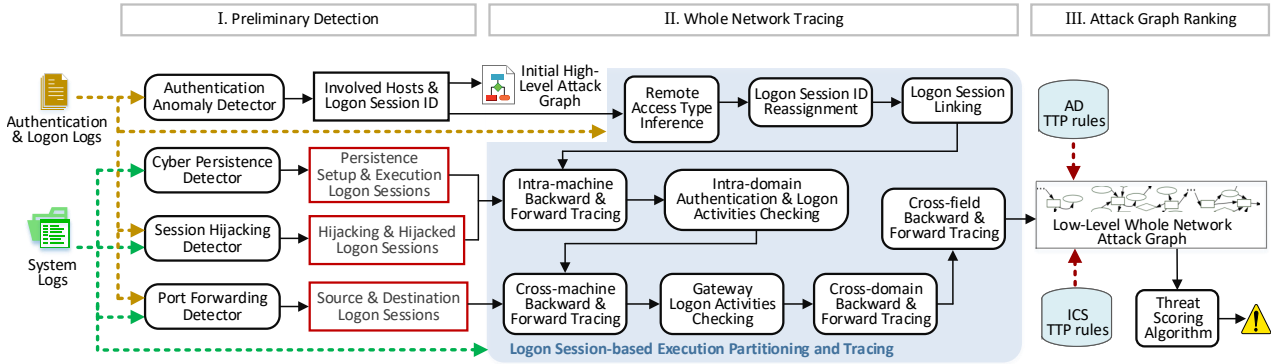



**Figure 9.1:** System overview of COMMANDER [P6].

## 9.2 COMMANDER Design

Our system COMMANDER follows a three-stage approach for efficient and accurate attack detection. We present COMMANDER's system architecture in Figure 9.1, and a formal description of COMMANDER's tracing and detection procedure in Algorithm 5. The first stage of COMMANDER serves as preliminary detection, and consists of four specialized detectors running in parallel: an authentication anomaly detector, a cyber persistence detector, a session hijacking detector and a port forwarding detector. While both the session hijacking detector and the port forwarding detector require system logs as well as authentication & logon logs, the authentication anomaly detector takes only authentication & logon logs as input, and the cyber persistence detector relies only on system logs.

The outcome of the authentication anomaly detector is used to initiate the whole network tracing in COMMANDER's stage two, whereas the other three detectors' detection results serve to counter evasion attacks and ensure robust and correct tracing in the second stage. In this stage, COMMANDER takes as input the identified host names, user names and their logon session ID for each anomaly from the authentication anomaly detector. It performs robust logon session-based execution partitioning and tracing to discover all involved logon sessions from each host the same identity, i.e., the same attacker or the same genuine user, has accessed. The whole network tracing ends, if no further logon session and host can be found, and produces a low-level whole network provenance graph for each anomaly spotted in COMMANDER's first stage.

In the final stage of COMMANDER, it matches the provenance graphs resulted from the whole network tracing against open-source TTP (Tactics, Techniques, Procedures) detection rules [74] for Active Directory discovery and credential access, and ICS TTP detection rules we developed for Siemens SIPROTEC and Hitachi Energy RTU500 industrial controllers. Then it ranks these attack graphs according to a threat scoring algorithm.

### 9.2.1 Preliminary Detection

In the following, we discuss the four specialized detectors in the preliminary detection stage of COMMANDER one by one. As the authentication anomaly

**Algorithm 5:** Robust Cross-Machine Tracing and Attack Detection [P6]

**Inputs :** System audit log events $\mathcal{E}$;
Authentication & logon events $\mathcal{A}$;
AD TTP rules $\mathcal{R}_{AD}$;
ICS TTP rules $\mathcal{R}_{ICS}$

**Output:** List $L_{<AG,TS>}$ of attack graph and its threat score pairs

1 **Function** GetAttackGraph($\mathcal{E}$, $\mathcal{A}$)
  /* Get a list of authentication & logon anomalies */
2  $L_{<\mathcal{A}\gamma,\mathcal{L}\gamma,\mathcal{T}\gamma>} \leftarrow$ DetectAuthenticationAnomaly($\mathcal{A}$)
  /* Get a list of persistence setup & execution logon session pairs */
3  $L_{<\mathcal{S}_{set},\mathcal{S}_{exe}>} \leftarrow$ DetectPersistence($\mathcal{E}$)
  /* Get a list of hijacking & hijacked logon session pairs */
4  $L_{<\mathcal{S}_{hg},\mathcal{S}_{hd}>} \leftarrow$ DetectSessionHijacking($\mathcal{E}$, $\mathcal{A}$)
  /* Get a list of post-port forwarding source & destination logon session pairs */
5  $L_{<\mathcal{S}_{src},\mathcal{S}_{des}>} \leftarrow$ DetectPortForwarding($\mathcal{E}$, $\mathcal{A}$)
6  $L_{<AG,TS>} \leftarrow$ null
7  **foreach** $(\mathcal{A}\gamma,\ \mathcal{L}\gamma,\ \mathcal{T}\gamma) \in L_{<\mathcal{A}\gamma,\mathcal{L}\gamma,\mathcal{T}\gamma>}$ **do**
8  $AG \leftarrow$ null
9  $HG \leftarrow$ CreateHighLevelGraph($\mathcal{A}\gamma$, $\mathcal{L}\gamma$)
10  $AG \leftarrow AG \cup HG$
11  $AccessType \leftarrow$ CheckRemoteAccessType($\mathcal{A}\gamma$, $\mathcal{E}$)
12  $SessionID \leftarrow$ ReassignSessionID($AccessType$, $\mathcal{A}\gamma$, $\mathcal{E}$)
13  $LinkedSessionID \leftarrow$ LinkSessions($SessionID$, $\mathcal{A}\gamma$, $\mathcal{E}$)
14  $(\mathcal{E}\alpha, G\alpha) \leftarrow$ TraverseBackward($SessionID$, $LinkedSessionID$, $\mathcal{E}$, $L_{<\mathcal{S}_{set},\mathcal{S}_{exe}>}$, $L_{<\mathcal{S}_{hg},\mathcal{S}_{hd}>}$, $L_{<\mathcal{S}_{src},\mathcal{S}_{des}>}$)
15  $(\mathcal{E}\kappa, G\kappa) \leftarrow$ TraverseForward($SessionID$, $LinkedSessionID$, $\mathcal{E}$, $L_{<\mathcal{S}_{set},\mathcal{S}_{exe}>}$, $L_{<\mathcal{S}_{hg},\mathcal{S}_{hd}>}$, $L_{<\mathcal{S}_{src},\mathcal{S}_{des}>}$)
16  $AG \leftarrow AG \cup G\alpha \cup G\kappa$
17  $\mathcal{E}\alpha \leftarrow \mathcal{E}\alpha \cup \mathcal{E}\kappa$
18  $\mathcal{A}\alpha \leftarrow$ CheckAuthenticationLogon($\mathcal{E}\alpha$, $\mathcal{A}$)
19  **if** $\mathcal{A}\alpha$ *is not null* **then**
20  **goto** 11
21  **end**
22  $TS \leftarrow$ GetThreatScore($AG$, $\mathcal{R}_{AD}$, $\mathcal{R}_{ICS}$)
23  $L_{<AG,TS>} \leftarrow L_{<AG,TS>} \cup (AG, TS)$
24  **end**
25  **return** $L_{<AG,TS>}$

26 **Function** DetectAuthenticationAnomaly($\mathcal{A}$)
27  $L_{<\mathcal{A}\gamma,\mathcal{L}\gamma,\mathcal{T}\gamma>} \leftarrow$ null
28  **foreach** $ae \in \mathcal{A}$ **do**
29  $\mathcal{A}_\alpha \leftarrow$ TraversalBackward($ae$)
30  $\mathcal{A}_\kappa \leftarrow$ TraversalForward($ae$)
31  $\mathcal{A}_\alpha \leftarrow \mathcal{A}_\alpha \cup \mathcal{A}_\kappa$
32  $(\mathcal{L}\alpha, \mathcal{T}\alpha) \leftarrow$ CheckAttackType($\mathcal{A}_\alpha$)
33  **if** $\mathcal{L}\alpha \in L_{<\mathcal{L}_{attack}>}$ **then**
34  $L_{<\mathcal{A}\gamma,\mathcal{L}\gamma,\mathcal{T}\gamma>} \leftarrow L_{<\mathcal{A}\gamma,\mathcal{L}\gamma,\mathcal{T}\gamma>} \cup (\mathcal{A}\alpha, \mathcal{L}\alpha, \mathcal{T}\alpha)$
35  **end**
36  **end**
37  **return** $L_{<\mathcal{A}\gamma,\mathcal{L}\gamma,\mathcal{T}\gamma>}$

detector and the cyber persistence detector are described in Chapter 8 and in Chapter 7, respectively, in detail, we discuss these two detectors only briefly in this chapter.

#### 9.2.1.1 Authentication Anomaly Detection

The authentication anomaly detector rests on a lightweight authentication anomaly detection model introduced in HADES. This model results from an extensive analysis of identity-based attacks in Active Directory, in which the complex network authentication process in Windows domain is exploited by attackers who succeed in stealing some form of credentials or credentials-related data on one machine, and using them to authenticate against a domain controller and hence accessing another machine.

In order to detect authentication anomalies, this module of COMMANDER inspects each authentication & logon log event, and performs backward tracing on each logon event, and backward & forward tracing on each authentication event (Algorithm 5 Lines 29-30). Note that the authentication process in Active Directory is a multiple-step process. The authentication & logon tracing's outcome is matched against the lightweight anomaly detection model to find out whether an authentication anomaly is evident and, if yes, to which attack type it belongs (Algorithm 5 Line 32). For each authentication anomaly, like HADES, COMMANDER produces a high-level attack graph including involved users and machines only (Algorithm 5 Line 9), marking the end of its stage one. Figure 9.2 presents an example of such high-level attack graphs at its top right corner. It shows that user Alice from the Exchange Server has used user Bob's password hash to authenticate against the Domain Controller, and then accessed the Data Server, following the Pass-the-Hash behavior.

#### 9.2.1.2 Cyber Persistence Detection

Understanding persistence attacks' nature, i.e., a successful persistence attack is always a two-part act consisting of the persistence setup phase and the persistence execution phase, is critical for persistence detection and hence correct provenance tracing. The persistence setup phase and the persistence execution phase often exist on different logon sessions, necessitating explicit and correct association of these logon sessions. Otherwise, broken links

between involved logon sessions controlled by the same attacker would be inevitable, ultimately resulting in incomplete cross-machine tracing.

We implement CPD as a preliminary detection module for COMMANDER, so that during COMMANDER's *whole network tracing*, persistence related logon sessions can be explicitly associated and hence broken links due to persistence attacks can be avoided. The cyber persistence detector first parses all system logs and matches them against a set of persistence setup detection rules introduced in Chapter 7 to create a *persistence setup table*. Next, it spots all processes with remote connections from those system logs, and performs backward tracing on these processes individually to create sub-graphs. Each sub-graph is then checked against a set of persistence execution detection rules, producing a *persistence execution table*.

Subsequently, the entries in the persistence setup table and persistence execution table are aligned with each other based on TTP labels, temporal order, and specific attributes. An alignment indicates a potential persistence attack. These alignments are fed to a false positive reduction algorithm introduced in CPD, which integrates key insights from real-world persistence attacks, for enhanced detection accuracy. The final outcome of the cyber persistence detector module in COMMANDER is a list of logon session pairs, each include a persistence setup logon session and a persistence execution logon session (Algorithm 5 Line 3).

#### 9.2.1.3 Session Hijacking Detection

COMMANDER's session hijacking detector module leverages two key insights: 1) a successful session hijacking will cause a logon event under hijacked user and a logoff event under hijacking user almost at the same time, and 2) in order to perform session hijacking, the attacker has to first conduct privilege escalation to receive the highest privilege, i.e. system privilege on Windows.

Algorithm 6 outlines the session hijacking detection procedure in COMMANDER, which starts from processing each logon event and then checking for a possibly related logoff event under a different user (Lines 3-4). After finding a related logoff event, it inspects the terminated logon session due to the logoff, and performs backward tracing from this logon session (Algorithm 6 Lines 6-7). Subsequently, it checks if the tracing result contains a privilege escalation

**Algorithm 6:** SESSION HIJACKING DETECTION [P6]

**Inputs :** System audit log events $\mathcal{E}$;
Logon events $\mathcal{O}$;
Logoff events $\mathcal{F}$
**Output :** List of hijacking & hijacked logon session pairs $L_{<\mathcal{S}_{hg},\mathcal{S}_{hd}>}$

1 $L_{<\mathcal{S}_{hg},\mathcal{S}_{hd}>} \leftarrow$ null
2 **foreach** $\mathcal{O}_\kappa \in \mathcal{O}$ **do**
/* Get the logon time, user and host of current logon event */
3 $(\mathcal{T}\kappa, \mathcal{U}\kappa, \mathcal{H}\kappa) \leftarrow$ GETATTRIBUTE($\mathcal{O}_\kappa$)
/* Check for a possibly related logoff event */
4 $\mathcal{F}_\gamma \leftarrow$ CHECKLOGOFF($\mathcal{F}$, $\mathcal{T}\kappa$, $\mathcal{U}\kappa$, $\mathcal{H}\kappa$)
5 **if** $\mathcal{F}_\gamma$ *is not null* **then**
6 $(SessionID, \mathcal{T}\gamma, \mathcal{U}\gamma) \leftarrow$ GETATTRIBUTE($\mathcal{F}_\gamma$)
7 $(\mathcal{E}\gamma, G\gamma) \leftarrow$ TRAVERSEBACKWARD($\mathcal{E}$, $\mathcal{H}\kappa$, $SessionID$, $\mathcal{U}\gamma$, $\mathcal{T}\gamma$)
/* Check for privilege escalation to the highest privilege */
8 $peh \leftarrow$ CHECKPRIVILEGEESCALATIONHST($\mathcal{E}\gamma$, $G\gamma$)
9 **if** $peh$ *is not null* **then**
10 $L_{<\mathcal{S}_{hg},\mathcal{S}_{hd}>} \leftarrow L_{<\mathcal{S}_{hg},\mathcal{S}_{hd}>} \cup (\mathcal{S}\gamma, \mathcal{S}\kappa)$
11 **end**
12 **end**
13 **end**
14 **return** $L_{<\mathcal{S}_{hg},\mathcal{S}_{hd}>}$

to the highest privilege (Algorithm 6 Line 8). With logon session-based execution partitioning and tracing, detecting privilege escalation is not only most accurate but also very efficient, as an operating system like Windows separates privileges using logon sessions. That is, given the OS integrity, COMMANDER's session hijacking detector only needs to check a few attributes, i.e., "Token Elevation Type" and "Integrity Level", in a process generation system event to detect a privilege escalation. Finally, the session hijacking detector outputs a list of logon session pairs consisting of a hijacking logon session and a hijacked logon session.

#### 9.2.1.4 Port Forwarding Detection

The port forwarding detector module in COMMANDER identifies port forwarding setup in each machine from system logs, and checks for occurrence of port forwarding-based logon from authentication & logon logs. First, it parses each system log event to get the process name, and checks if it belongs to the list of port forwarding-related processes, e.g., `plink`, `ssh`, `netsh` (Algorithm 7 Lines 3-4). Next, it parses the command line arguments of that process to

**Algorithm 7:** PORT FORWARDING DETECTION [P6]

**Inputs :** System audit log events $\mathcal{E}$;
Authentication & logon events $\mathcal{A}$;
List of port forwarding-related processes $L_{<\mathcal{P}>}$
**Output:** List of source & destination logon session pairs $L_{<\mathcal{S}_{src},\mathcal{S}_{des}>}$

1 $L_{<\mathcal{S}_{src},\mathcal{S}_{des}>} \leftarrow$ null
2 **foreach** $\mathcal{E}_\kappa \in \mathcal{E}$ **do**
  /* Get process name, host name, and timestamp of current event */
3  $(\mathcal{P}_\kappa, \mathcal{H}\kappa, \mathcal{T}\kappa) \leftarrow$ GETSUBJECT($\mathcal{E}_\kappa$)
4  **if** $\mathcal{P}_\kappa \in L_{<\mathcal{P}>}$ **then**
    /* Parse command line arguments */
5  $(IP_{src}, Port_{src}, IP_{det}, Port_{det}) \leftarrow$ CHECKCMD($\mathcal{P}_\kappa$)
6  **if** $Port_{src}$ *is not null* **then**
7  $po \leftarrow$ CHECKPORTOPENING($Port_{src}$, $\mathcal{E}$)
8  $IP_{pro} \leftarrow$ GETIP($\mathcal{H}\kappa$)
9  **if** $po$ *is not null* **then**
10  $(\mathcal{A}_\gamma) \leftarrow$ CHECKLOGON($\mathcal{A}$, $\mathcal{T}\kappa$, $IP_{pro}$, $IP_{det}$, $Port_{det}$)
11  **foreach** $ae \in \mathcal{A}_\gamma$ **do**
12  $(\mathcal{S}\gamma, \mathcal{S}\omega) \leftarrow$ TRAVERSALBACKWARD($ae$, $IP_{src}$)
13  $L_{<\mathcal{S}_{src},\mathcal{S}_{des}>} \leftarrow L_{<\mathcal{S}_{src},\mathcal{S}_{des}>} \cup (\mathcal{S}\gamma, \mathcal{S}\omega)$
14  **end**
15  **end**
16  **end**
17  **end**
18 **end**
19 **return** $L_{<\mathcal{S}_{src},\mathcal{S}_{des}>}$

determine the source port & IP and destination port & IP for the port forwarding (Algorithm 7 Line 5). If a source port can be identified, it checks in system logs if this port was opened through the host-based firewall, and the IP address of this proxy machine (Algorithm 7 Lines 7-8). Subsequently, it parses all logon events on the destination host that have happened after the timestamp of the port forwarding setup event and have the IP address of the proxy machine as source IP (Algorithm 7 Line 10). For each found logon event, it performs backward tracing with the corrected source IP address to find the source logon session, and then appends the source & destination logon session pair to a list.

### 9.2.2 Whole Network Tracing

**Industrial-sector organization.** An industrial-sector organization typically has an enterprise/IT network and an ICS/OT (Operational Technology) network that are separated from each other [160]. Communications between

these networks are often enabled by a gateway/jump server. To reduce the risk, the enterprise network and the ICS network have separate identity and access management (IAM) systems [41], [161]. IAM systems are ubiquitous in networks of large organizations. (Microsoft) Active Directory is the most popular IAM system, with more than two-thirds of the market share and 90% of Fortune 1000 companies relying on it [40], [45], [214]. Active Directory is popular not only in IT networks, but also in OT networks. In fact, OT vendors, e.g., Schneider Electric, shifted their OT products' OS from Unix-based ones to Windows NT long time ago in order to better make use of Active Directory functionalities like Kerberos Authentication and Group Managed Service Accounts [215]. Although our implementation and evaluation of COMMANDER rest on Active Directory, the concept for robust whole network tracing and efficient attack detection is transferable to networks employing another IAM system.

**Attacks on industrial-sector organizations.** As shown in [59], during an attack against an industrial-sector organization, a complete attack path includes *multiple cross-machine movements* necessary for APT attackers to achieve their final objectives. Often, attackers first gain an initial foothold on a machine in the IT network through phishing or binary exploitation. From the initially accessed machine, attackers routinely perform Active Directory discovery to locate more machines in the network. Moreover, attackers commonly steal credentials, e.g., cached authentication tickets, password hashes, or even clear text passwords, on each accessed machine. Leveraging stolen credentials is the most popular way for attackers to *move across a target network* [37]–[41]. The pivotal access on a gateway server enables attackers to reach machines in the OT network. As the objective of attacks against industrial-sector organizations is impairing physical processes, the final target of the attackers is typically Engineering Workstations, through which field controllers are programmed and configurated, and field controllers, like PLC (Programmable Logic Controller), RTU (Remote Terminal Unit) and IED (Intelligent Electronic Device), through which physical processes are controlled.

In its stage two, COMMANDER performs whole network tracing, which improves HADES's logon session-based tracing with enhanced robustness against evasion attacks, and extends it to trace also cross domains inside an industrial-sector organization. This stage starts with processing the data delivered from the authentication anomaly detector, i.e., identified host names, user names

and logon session ID. First, it infers the remote access type, in which it pulls both related authentication & logon events and system events, and extracts a list of attributes to analyze the context and then classify the remote access type (Algorithm 5 Line 11).

Next, it performs logon session ID reassignment and logon session linking according to the remote access type (Algorithm 5 Lines 12-13). Logon session ID reassignment is necessary for several remote access types, e.g., RDP (Remote Desktop Protocol) and Powershell-Remoting, as the system activities performed on a new logon session are recorded under an existing logon session ID. With logon session linking, a user logon session is coupled with the system logon session directly responsible for the creation of that user logon session. Note that a system logon session is normally a long-running session accommodating numerous system activities. The logon session linking module selects only a fraction of these system activities, which are related to the user session creation, to parse into the provenance graph.

Subsequently, Commander proceeds with intra-machine backward & forward tracing to find all involved logon sessions and system activities inside them (Algorithm 5 Lines 14-15). This step automatically spots privilege escalation, as system activities run with different privileges are contained in different logon sessions and labeled with different session ID. A process's privilege is tied to the security context of the logon session in which the process is running. During normal process creation, both child and parent processes run in the same logon session. But, during a privilege escalation, when a process is spawned from another process with lower privilege, the spawned process runs in a new logon session. Unlike Hades, Commander checks in this step also if current logon sessions are involved in a persistence attack or a session hijacking attack by inspecting the list of persistence setup & execution logon session pairs, and the list of hijacking & hijacked logon session pairs, passed from the cyber persistence detector and the session hijacking detector, respectively. If it finds a match, it connects the related logon sessions with a corresponding edge, and continues the intra-machine backward or forward tracing on the newly joined logon session.

Afterwards, Commander examines the authentication & logon logs from the same domain to determine the source machine & logon session of current logon sessions, i.e., cross-machine backward tracing, and to identify logon sessions inside any domain-joined machine initiated from the current machine, i.e., cross-machine forward tracing. During this step, Commander also

checks if the traced logon sessions are involved in a port forwarding attack by inspecting the list of source & destination logon session pairs output from the port forwarding detector, and attaches any newly found logon sessions to existing ones. At the end of intra-domain tracing, COMMANDER checks logon activities on the gateway server. If any logon initiated from a machine in the IT domain is found, and that machine is involved in the current tracing, COMMANDER begins the cross-domain tracing. COMMANDER's performance to trace cross domains inside an organization benefits from the fact that an industrial-sector organization's network is generally segmented into an IT domain network and an OT domain network, and the connection of these two domains is done via a gateway/jump server.

Following the cross-domain tracing, COMMANDER checks if the Engineering Workstation is involved in the tracing. If yes, it then examines whether there are logon activities on the field controllers coming from a logon session in the Engineering Workstation. For each logon event, it parses all log events between that logon and the corresponding logoff from the Engineering Workstation. Field controllers typically have a single-user single-task embedded OS. Unlike general-purpose OS, embedded OS do not implement the concept of logon session, meaning that system activities cannot be sorted by logon session ID. However, thanks to its single-user restriction, it is straightforward to correctly link a specific logon session on the Engineering Workstation with only system activities on a field controller that are indeed caused by commands from that session.

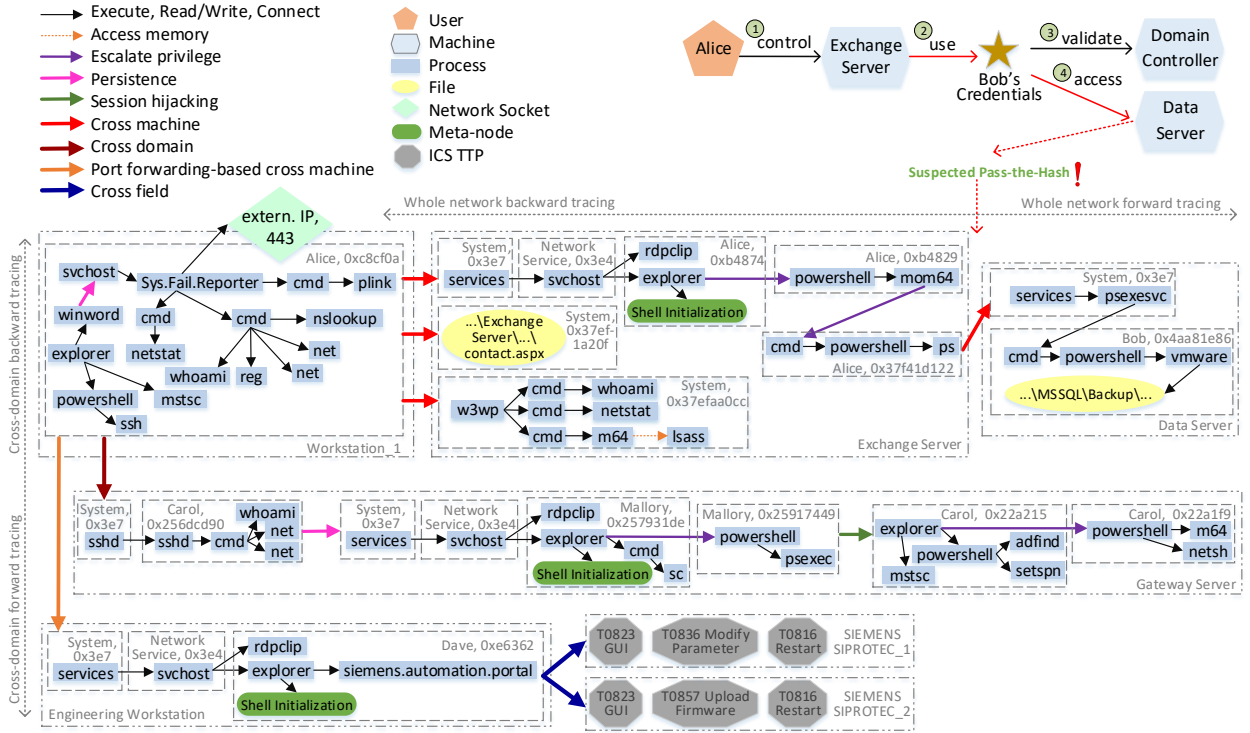



**Figure 9.2:** An attack graph created by COMMANDER on the AVIATOR-Oilrig-extended dataset [P6]. Note that the bounding boxes with a session ID label or host name are manually added for better readability, this information is encoded in nodes in the original attack graph.

Figure 9.2 renders the result of whole network tracing for the attack in the AVIATOR-Oilrig-extended dataset, unraveling the attacker's traversal inside the target organization. COMMANDER first creates an initial high-level attack graph including only high-level entities, i.e., users and hosts, after the authentication anomaly detector identifies a potential Pass-the-Hash attack. In Pass-the-Hash attacks [139], attackers leverage stolen password hashes and choose NTLM authentication process over the default Active Directory authentication process Kerberos. Then it performs system-level forward tracing inside the specific logon session under user Bob in the accessed host Data Server, and system-level forward & backward tracing inside the logon session of Alice in the accessing host Exchange Server. The remote access type on the Data Server is identified as PsExec [216], for which session ID reassignment is not required. User Bob's logon session (`0x4aa81e86`) is linked to the System logon session (`0x3e7`), as the server process `psexesvc` runs in the System logon session, which receives remote commands from the Exchange Server, and the remote commands run in Bob's logon session. The whole network forward tracing from the Data Server leads to no further logon session on another machine, as the attacker found the targeted data and did not move forward from the Data Server.

In contrast, the whole network backward tracing from the Exchange Server discloses that the current logon sessions (`0xb4874`, `0xb4829`, `0xb37f41d122`) on it belong to a RDP session that was initiated from another logon session of user Alice inside another machine Workstation_1. Besides, it spots a privilege escalation from unprivileged user to Administrator, i.e., from `0xb4874` to `0xb4829`, and a further privilege escalation from Administrator to System, i.e., from `0xb4829` to `0xb37f41d122`. Subsequently, it proceeds with the backward & forward tracing from the logon session of Alice (`0xc8cf0a`) in the Workstation_1. While backward tracing from the Workstation_1 leads to no further domain-joined machine, but rather an external IP address that belongs to the C2 server run by the attacker, forward tracing from the Workstation_1 unveils multiple logon sessions on several machines[1].

Among the newly found logon sessions, two sessions are on the Exchange Server. These two sessions predate other sessions on the Exchange Server,

[1] Note that backward tracing can lead to no more than one logon session, whereas forwarding tracing may lead to multiple logon sessions, as one may remotely access several other machines from one machine.

with one session containing system activities related to credential access, contributing to the lateral movement to the Data Server. Unlike HADES, COMMANDER also performs cross-domain tracing inside the same organization, resulting in identifying a logon session in the Gateway Server located in the OT domain. Besides, each time COMMANDER discovers a new logon session during tracing, it checks against the logon session pairs identified by its cyber persistence detector, session hijacking detector and port forwarding detector, respectively, to find out whether this logon session is involved in stealthy attacks that could impede accurate tracing. By doing so, COMMANDER discovers a few more logon sessions in the Gateway Server and in the Engineering Workstation. The logon session in the Engineering Workstation under user Dave finally leads to logon sessions on two Siemens field controllers.

Because the first logon session of user Carol (`0x256dcd90`) is identified as a persistence setup logon session by the cyber persistence detector, a persistence-based cross-session edge, named as pseudo-edge in CPD [18], is created to connect it to the persistence execution logon session (`0x3e7`). Without the cyber persistence detector, forward tracing from Carol's first logon session would end up finding no further logon session and stopping the tracing. During the persistence attack step, the attacker has created a new local user Mallory on the Gateway Server. This persistence technique [217] provides the attacker the ability to access the desktop environment on the Gateway Server. Note that the first access to the Gateway Server was done via SSH and through public key authentication. That is, without knowing user Carol's password, the attacker was able to log into the Gateway Sever as Carol using a private key stored in the Workstation_1.

Following the persistence execution, the attacker hijacked user Carol's RDP session and hence the logon sessions (`0x22a215`, `0x22a1f9`) inside it. From these logon sessions, the attacker was able to conduct Active Directory discovery to find more machines in the OT domain, as the user Carol is a domain user and the attacker-created user Mallory is not[2]. Besides, the attacker has set up a port forwarding in the session with higher privilege (`0x22a1f9`). En-

[2] Note that only domain administrators and specifically delegated users can create new domain accounts. Local users are restricted from accessing domain information. Depending on the remote access type, a domain user's logon session's security context may not be sufficient for accessing domain information, a restriction implemented on Windows. That is, from the SSH-enabled logon session (`0x256dcd90`) of the domain user Carol, the attacker was unable to conduct Active Directory discovery.

abled by the port forwarding, the attacker was able to access the Engineering Workstation in the OT domain directly from the Workstation_1 in the IT domain. Thanks to the port forwarding detector, the link to the Engineering Workstation can be discovered during the tracing. Further, cross-field tracing is initiated from the Engineering Workstation. Due to the coarse granularity of system logs on field devices, it is not feasible to create links between the system activities inside a device. Hence, in the attack graph, we use more abstract entities, i.e., matched ICS TTP, and cluster and link them to the logon session on the Engineering Workstation from which the corresponding commands originate.

### 9.2.3 Attack Graph Ranking

**Algorithm 8:** COMMANDER'S THREAT SCORING [P6]

**Inputs :** Attack graph $AG$;
AD TTP rules $\mathcal{R}_{AD}$;
ICS TTP rules $\mathcal{R}_{ICS}$
**Output:** Threat score $TS$ of attack graph $AG$

$L_{<TS_{sub}>} \leftarrow$ null
**foreach** $(CrossMachineEdge, \mathcal{T}\omega) \in AG$ **do**
    /* Get a list of discovery techniques and frequency */
    $L_{<tec_d, freq>} \leftarrow$ CHECKADDISCOVERY($AG$, $\mathcal{T}\omega$, $\mathcal{R}_{AD}$)
    $L_{<tec_{ca}, freq>} \leftarrow$ CHECKCREDENTIALACCESS($AG$, $\mathcal{T}\omega$, $\mathcal{R}_{AD}$)
    $L_{<pe>} \leftarrow$ CHECKPRIVILEGEESCALATION($AG$)
    $L_{<per>} \leftarrow$ CHECKPERSISTENCE($AG$)
    $L_{<sh>} \leftarrow$ CHECKSESSIONHIJACKING($AG$)
    /* Check if current cross-machine edge is port forwarding based */
    $pf \leftarrow$ CHECKPORTFORWARDING($CrossMachineEdge$)
    $cd \leftarrow$ CHECKCROSSDOMAIN($CrossMachineEdge$)
    /* Get a list of ICS attack techniques and frequency */
    $L_{<tec_{ics}, freq>} \leftarrow$ CHECKICSTTP($AG$, $\mathcal{R}_{ICS}$)
    $TS_{sub} \leftarrow$ CALCULATESCORE($L_{<tec_d, freq>}$, $L_{<tec_{ca}, freq>}$, $L_{<pe>}$, $L_{<per>}$, $L_{<sh>}$, $pf$, $cd$, $L_{<tec_{ics}, freq>}$)
    $L_{<TS_{sub}>} \leftarrow L_{<TS_{sub}>} \cup TS_{sub}$
**end**
$TS \leftarrow$ SUMSCORE($L_{<TS_{sub}>}$)
**return** $TS$

COMMANDER's threat scoring scheme (Algorithm 8) in its final stage builds on HADES's, resting on two key insights introduced in HADES: 1) long-running attacks against a large organization follow a rigid pattern after the initial access: Active Directory discovery, credential access, lateral movement and privilege

escalation, 2) a successful credential access is essential for the attacker to proceed with the attack, and the most observable result of an ongoing attack is lateral movement. For each whole network attack graph output from COMMANDER's stage two, COMMANDER calculates a threat score. By ranking attack graphs via threat score, COMMANDER presents security analysts with the most likely true attack graphs first for further investigation.

COMMANDER extends HADES's threat scoring scheme by also considering the occurrence of 1) persistence, session hijacking and port forwarding attacks (Algorithm 8 Lines 6-8), 2) cross-domain activities (Line 9), and 3) ICS TTP (Line 10) in an attack graph for score calculation. That is, for each potential lateral movement, i.e., a cross-machine edge in an attack graph, it checks not only the variety and frequency of credential access techniques and Active Directory discovery techniques used in the source host, but also the presence of persistence and session hijacking attacks in both source and destination hosts, whether this is a port forwarding-enabled or a cross-domain lateral movement, and whether the destination host is a field controller on which ICS TTP are identified. Each occurrence of these indicators increases the threat score.

More formally, the threat score is determined by the following two equations, of which Equation 9.1 calculates the threat score for each cross-machine edge, and Equation 9.2 accumulates the threat scores passed from Equation 9.1, and yields the final score.

$$TS_E = \sum_{i=1}^{n} (Freq(teq_i) \times Var(tac_i))^{w_i} \tag{9.1}$$

where $n$ denotes the number of tactics involved, e.g., credential access or persistence. $Freq(teq_i)$ denotes the accumulated execution frequency of techniques in a given tactic, and $Var(tac_i)$ denotes the number of techniques being applied for the given tactic. $w_i$ is a weighting factor for each tactic, with credential access having the highest weight due to its importance as per the second insight.

$$TS_G = \sum_{i=1}^{n} (TS_i \times (DA + 1) \times Criticality) \tag{9.2}$$

where $n$ denotes the number of cross-machine edges found in a given attack graph, and $TS_i$ denotes the threat score assigned for each cross-machine edge from Equation 9.1. $DA$ indicates whether credentials of domain administrators are involved in the cross-machine activities, and has the value 0 or 1. Different values of $Criticality$ are given to different attack types, in which the Golden-Ticket attack type receives the highest value, as it implies a compromise of an entire domain. Besides, it scales up $Criticality$ accordingly, if a cross-machine edge is port forwarding enabled, or connects two machines from two different domains, i.e., cross-domain, or is tied to a field controller on which ICS TTP are identified, i.e., cross-field.

## 9.3 Implementation

We implemented a prototype of Commander in Python, and deployed it on a 64-bit Ubuntu 22.04 OS with 256 GB of RAM and a 32-core processor. This machine also hosts Elasticsearch [169], to which Commander interfaces via EQL [175], a query language designed for security purposes. Like Hades, Commander operates as an on-demand tracing system for enhanced efficiency. That is, it starts the whole network tracing, only after an authentication anomaly is identified by the authentication anomaly detector in the preliminary detection stage. During the tracing, it searches for relevant events to process via Elasticsearch, which provides scalable and near real-time search for log data investigation. Moreover, we use Python NetworkX [176] for creating provenance graphs on demand, and PyVis [177] for graph visualization.

Whereas authentication logs for each domain are collected from the corresponding domain controller, logon logs are collected from each accessed domain-joined machine. System logs are collected from all machines. On Windows, we collect Windows Security [155] logs and Sysmon [84] logs, while on Linux we collect Auditd [85] logs. These logs are all industry-standard logs, boosting Commander's applicability in industry.

## 9.4 Evaluation

In this section, we evaluate COMMANDER's performance on two extended MITRE emulation plans. In particular, we compare COMMANDER's detection accuracy with prior detection systems, showcase COMMANDER's response time, and analyze attack graphs produced by COMMANDER.

### 9.4.1 Datasets & Experimental Setup

**Public Datasets.** Attack scenarios in public datasets taken for evaluation of prior PIDS often lack not only a persistence attack step, but also a lateral movement attack step, even though these steps are omnipresent in real-world APT attacks. For instance, the DARPA-E3 dataset [69], which is one the most used datasets for PIDS evaluation, contains neither a persistence attack step, nor a lateral movement trace. Although a subset of the DARPA-E5 dataset [70] includes two Windows-based persistence attack instances, attacks in this dataset are confined in individual machines that are not joined to a domain. The more recent DARPA-OpTC dataset [71] is the only public dataset emulating APT attacks in a realistic enterprise environment with domain-joined machines. However, this dataset contains only system logs on domain-joined machines, but no logs from the domain controller. Authentication logs, which are recorded only on domain controllers, are critical for not only three out of four COMMANDER's preliminary detectors, but also the whole network tracing. Further, logs in the DARPA-OpTC dataset lack a logon session ID, making it even more infeasible to evaluate COMMANDER on this dataset.

**MITRE Attack Emulation Plans.** In contrast, at least two persistence attack techniques are included in each of MITRE's eleven full emulation plans [46], whereas Active Directory-based lateral movement is present in nine out of eleven emulation plans. These emulation plans are based on the observation of past real-world APT attacks, and target enterprise networks with more sophisticated, cross-machine attacks than most public datasets, offering greater authenticity. However, MITRE has not published any corresponding datasets. Therefore, we resort to stringently implementing MITRE's emulation plans according to instructions published by MITRE.

**Extended MITRE Emulation Plans.** Unfortunately, none of MITRE's eleven emulation plans includes attack steps on ICS. Hence we extended two

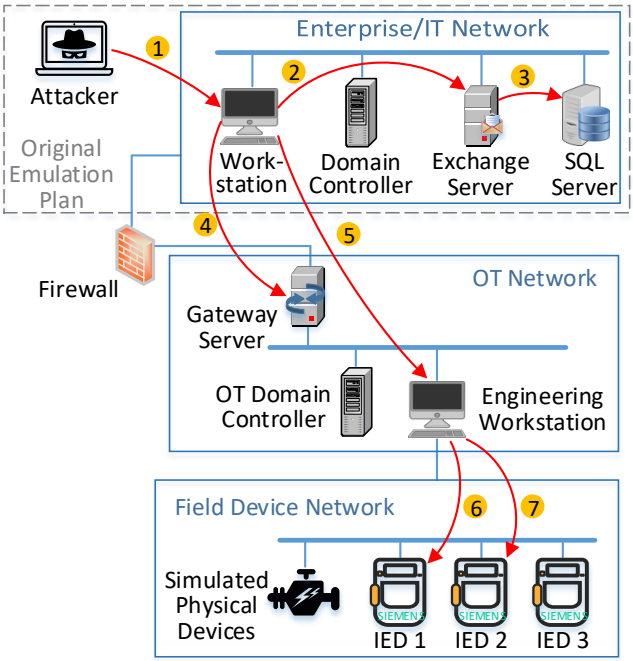



**Figure 9.3:** Extended Oilrig emulation plan [P6]. Edges in red denote initial access or lateral movement conducted by the attacker. These edges are numbered in chronological order.

MITRE emulation plans, i.e, Sandworm [87], Oilrig [89], from APT groups most known for their attacks on ICS in real world. Whereas Sandworm is responsible for the attacks against Ukrainian electrical companies in 2015, 2016 and 2022 [54], [149]–[151], Oilrig has targeted energy and chemical sectors in Middle Eastern in 2017 [152]–[154]. We expanded the emulation infrastructures to include an industrial control network consisting of a gateway server, an OT domain controller, a typical Windows-based Engineering Workstation and several field devices. Our extended emulation plans are demonstrated in Figure 9.3 and Figure 9.4, in which the original emulation plans are shown in a dashed bounding box, respectively. We evaluate COMMANDER on the datasets resulted from the extended emulation plans, named as AVIATOR-Oilrig-extended and AVIATOR-Sandworm-extended, respectively.

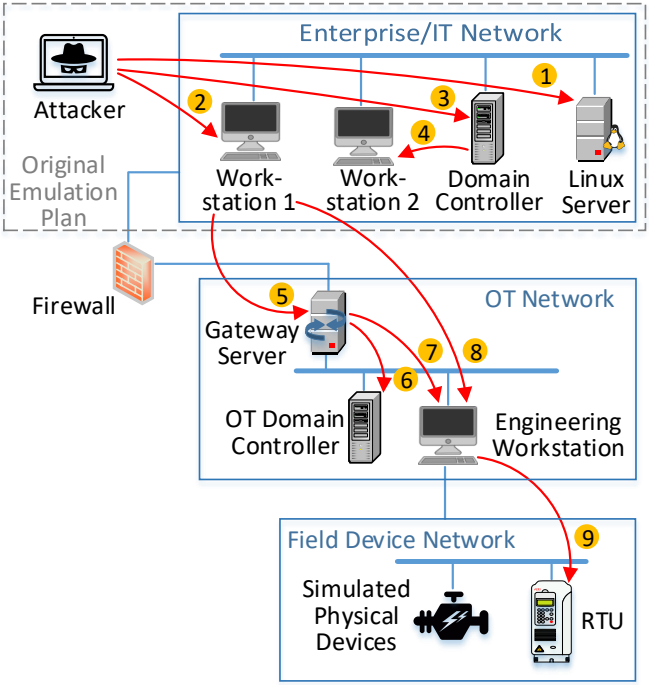



**Figure 9.4:** Extended Sandworm emulation plan [P6].

**Table 9.1:** Overview of COMMANDER's evaluation datasets [P6].

| Dataset | Target Host Number | Ground Truth Attack | Target Host OS | Data Size | Event Number |
|---|---|---|---|---|---|
| **AVIATOR-Oilrig-extended** | 10 | Pass-the-Hash | Windows Embedded OS | 566GB | 360M |
| **AVIATOR-Sandworm-extended** | 8 | Golden-Ticket | Windows Linux Embedded OS | 610GB | 479M |

Table 9.1 gives an overview of these datasets. Note that these datasets are included in the AVIATOR dataset as subsets.

**Table 9.2:** Comparison of Commander and prior detection systems [P6].

| Dataset[a] | Commander | | | | Elastic | | Chronicle | | Sigma | | Cad | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Preliminary[b] | | Final | | | | | | | | | |
| | FP | FN | FP | FN | FP | FN | FP | FN | FP | FN | FP | FN |
| Aviator-Oilrig-extended | 105 | 0 | 0 | 0 | 208 | 0 | 50 | 1 | 415 | 0 | 0 | 1 |
| Aviator-Sandworm-extended | 59 | 0 | 0 | 0 | 196 | 1 | 31 | 1 | 722 | 0 | 0 | 0 |

[a] Note that each dataset has only one true positive (TP=1).
[b] Commander's preliminary result refers to the detection result after applying only the lightweight anomaly authentication model, while Commander's final result refers to the detection result after also performing whole network tracing and threat score calculation.

### 9.4.2 Commander vs. Prior Detection Systems

To compare Commander with prior detection systems, including open-source detection rules and a commercial Active Directory attack detector, we first extracted all detection rules for Active Directory-based attacks from the most well-known rule repositories, i.e., Elastic [146], Sigma [74], Google Chronicle [75], and installed a commercial Active Directory attack detector. Both the EQL queries converted from the detection rules and the commercial attack detector are evaluated alongside Commander on the datasets. The commercial attack detector is referred to as Cad in this work, since we did not get the permission to reveal the name of the security vendor, which is considered as a significant security solution provider in the security market overviews [212], [213]. We compare the detection results in Table 9.2.

Similar to Hades's evaluation results, Table 9.2 shows that open-source detection rules generated more false positives (FP) than a full-fledged attack detector like Commander and Cad, and missed some true attacks. More specifically, Elastic produced more false positives than Commander's preliminary detection stage on both datasets, while missing the true attack in the Aviator-Sandworm-extended dataset. Although Chronicle has a lower false positive rate on both datasets, it did not detect any true attack. In contrast, Sigma identified the true attacks in both datasets, however at the expense of

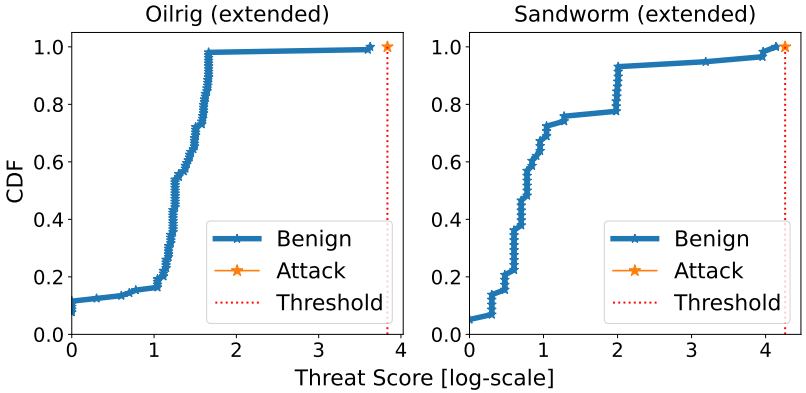



**Figure 9.5:** CDF of threat score for false and true alerts [P6].

producing much more false positives. Open-source detection rules' overly high false positive rate is well demonstrated and discussed in [18], [19].

Contrary to open-source detection rules, a dedicated attack detector like Commander and Cad associates events related to each other, and analyzes the context, moving beyond inspecting individual log events independently. This context-awareness can significantly lower the false positive rate, as demonstrated by the comparison between the detection results of Commander's preliminary detection stage and final detection stage, i.e., after the whole network tracing. Note that most attack steps involved in stealthy APT attacks are based on misusing legitimate system infrastructures and services, which are used intensively by genuine users as well. The prevalence of "indicative" system activities in benign operations helps attackers easily blend each of their attack steps into the noise caused by genuine users. However, with precise tracing, the system activities of individual attack steps can be linked into a provenance graph, as they are causally related.

Comparing to a provenance graph built from a genuine user's all system activities, a provenance graph containing the whole of an attacker's system activities typically includes more "indicative" activities and outlines a logical order, e.g., lateral movement after credential access, underpinning threat scoring schemes in PIDS like [18], [19], [24]–[26] and Commander. Often the longer the attack chain is, the more obvious the attack pattern is and thus the more likely the attack provenance graph is ranked higher than benign

ones. In comparison to HADES, which has assigned the third-highest score to the true attacks in two datasets, COMMANDER has produced no false positive by ranking the true attack in each dataset the highest, thanks to the longer attack chain in the extended emulation plans and COMMANDER's robust whole network tracing ability. Note that COMMANDER's final detection results in Table 9.2 are based on the true attack threat score threshold in Figure 9.5, which shows the cumulative distribution function of threat scores for benign and attack provenance graphs.

Further, the detection result comparison between CAD and COMMANDER's final stage in Table 9.2 shows that COMMANDER outperforms CAD by accurately detecting the true attacks in both datasets, as a result of COMMANDER's more advanced tracing module than CAD's. By investigating attack graphs output from CAD, we find that these are more abstract graphs, whose graph granularity is only comparable to the initial high-level attack graphs produced at the end of COMMANDER's preliminary detection stage. Consulting on the documentation, which was provided by the vendor and includes a guideline for CAD's logging configuration and detection logic for various kinds of attacks, reveals that CAD does not collect system log events, but rather a subset of authentication & logon event types. This leads to limited visibility into intra-machine system activities, which we deem indispensable for accurate tracing and attack detection. Our engagement with the vendor over weeks confirmed that CAD's detection on Pass-the-Hash attacks has known issues, contributing to CAD's false negative (FN) on the AVIATOR-Oilrig-extended dataset.

### 9.4.3 Response Time

Figure 9.6 shows the cumulative distribution function of COMMANDER's response time on each dataset. COMMANDER's response time is measured per attack graph, i.e., the time it takes to output a single attack graph with calculated threat score. For each attack graph, the time spent on COMMANDER's stage two & three, i.e., whole network tracing and threat score assignment, vary between only 7 seconds and 32 seconds on the AVIATOR-Oilrig-extended dataset, and between only 11 seconds and 52 seconds on the AVIATOR-Sandworm-extended dataset. However, the absolute response time to output an attack graph ranges from 103 seconds to 128 seconds on the AVIATOR-Oilrig-extended dataset, and from 586 seconds to 627 seconds on the AVIATOR-Sandworm-extended dataset.

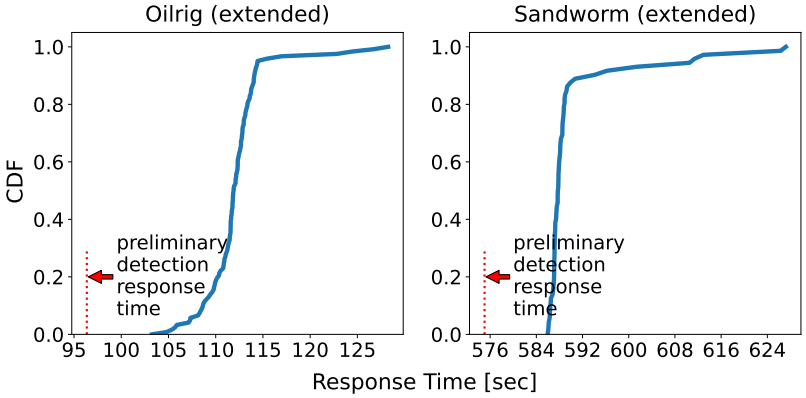



**Figure 9.6:** CDF of response time of COMMANDER [P6].

This is due to a bottleneck resulted from COMMANDER's system design, i.e., COMMANDER's stage two only starts after all preliminary detectors in its stage one finished.

The preliminary detection response time (marked with a dotted line pointed by a red arrow in Figure 9.6) takes the largest of response times of the four preliminary detectors running in parallel. On both datasets, the cyber persistence detector has the longest response time among the four detectors, reaching 96 seconds on the AVIATOR-Oilrig-extended dataset and 575 seconds on the AVIATOR-Sandworm-extended dataset. The response time of the cyber persistence detector, as a preliminary detector, is a value accumulated from times for processing all suspected persistence instances identified on each host. That is, this detector's longer response time on the AVIATOR-Sandworm-extended dataset results from the fact that it has picked out noticeably more suspected persistence instances to process on this dataset, including false-positive ones.

### 9.4.4 Reconstructed Attack Graphs

Whereas the reconstructed attack graph for the attack in the AVIATOR-Oilrig-extended dataset (Figure 9.2) is already interpreted in Section 9.2.2, we discuss in this section the true positive attack graph from the AVIATOR-Sandworm-extended dataset in Figure 9.7. The tracing on this attack starts from the

authentication anomaly mimicking a Golden-Ticket behavior in the OT domain. In Golden-Ticket attacks [202], attackers use stolen credentials of the `krbtgt` account from a domain controller for authentication, and can impersonate any domain account, essentially having the ability to move laterally to any domain-joined machine.

After identifying the Gateway Server as the source host, from which the attacker has used the stolen credentials, and the Engineering Workstation as the destination host to which the attacker has moved, COMMANDER performs forward tracing from the involved logon session under user Dave (`0x2b8d50717`) on the Engineering Workstation, and forward & backward tracing from the involved System logon session (`0x3e7`) on the Gateway Server. The forward tracing from the session `0x2b8d50717` leads to no further machine or logon session, while the forward tracing from the session (`0x3e7`) on the Gateway Server exposes a user logon session (`0x2b65c94e`) belonging to the domain administrator on the OT domain controller.

As COMMANDER also checks the list of persistence setup & execution logon session pairs during the tracing, and correctly identified the session (`0x3e7`) on the Gateway Server as a persistence execution logon session leveraging WMI[3], the backward tracing from this session results in a logon session under user Carol (`0x3c128d94`) on the Gateway Server, i.e., the corresponding persistence setup logon session. The link between these two sessions can only be found by a persistence detector. The cyber persistence detector in COMMANDER's stage one correctly identifies this as a WMI persistence instance [201]. Otherwise the backward tracing would lead to nowhere, but a overly premature termination of the whole network tracing and largely incomplete attack graph, like the one HADES would output in this case.

[3] In Section 7.4.4, we show a graph of a persistence attack exploiting WMI, while in Table 7.1, we present insights in detecting persistence attacks exploiting WMI.

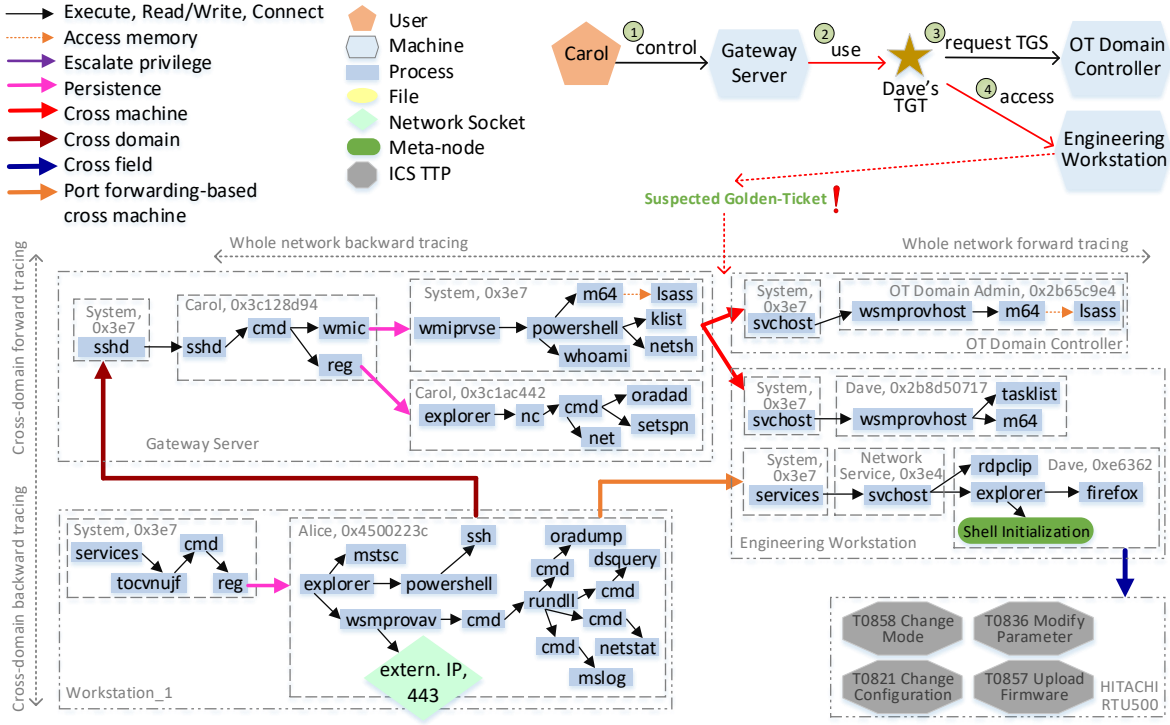



**Figure 9.7:** An attack graph created by COMMANDER on the AVIATOR-Sandworm-extended dataset [P6].

Further, another logon session of user Carol (`0x3c1ac442`) is spotted during the forward tracing from the newly found session (`0x3c128d94`), as the session (`0x3c1ac442`) is a persistence execution logon session of another persistence instance Registry Run keys [132]. That is, the attacker has executed two persistence techniques on the same logon session `0x3c128d94`, so that every time the machine restarts, the attacker gets a call back from a System logon session, or every time the user Carol logs in, the attacker receives a call back from Carol's logon session like `0x3c1ac442`. On the System logon session (`0x3e7`), the attacker conducted credential access without performing privilege escalation first, and port forwarding & port opening through host-based firewall, since this logon session is always assigned with the highest privilege. On the user logon session `0x3c1ac442`, the attacker executed several Active Directory discovery techniques with the programs `net`, `setspn` and `oradad`. Note that chronologically speaking, instead of in the tracing order, this happened before the lateral movement to the OT Domain Controller and the Engineering Workstation.

Next, backward tracing from the session `0x3c128d94` reveals a logon session of user Alice (`0x4500223c`) on the Workstation_1 from the IT domain. Further backward tracing from Alice's session `0x4500223c` discloses that this session is a persistence execution logon session, and hence is linked to the corresponding persistence setup logon session, i.e., `0x3e7` on the Workstation_1. Forward tracing from the session `0x4500223c` leads to another user logon session of Dave (`0xe6362`) on the Engineering Workstation in the OT domain, thanks to the list of source & destination logon session pairs output from the port forwarding detector from COMMANDER's stage one. Without this list, the tracing would end prematurely at this step. Finally, forward tracing from Dave's logon session `0xe6362` results in a session on the field controller Hitachi RTU500 and several activities in it that can be mapped to ICS TTP, e.g., modify parameters, further increasing the threat score of this attack graph.

## 9.5 Limitations & Discussion

Although COMMANDER has successfully detected the true attacks in both datasets, the reconstructed attack graph from the AVIATOR-Sandworm-extended dataset is not entirely complete. In practice, a PIDS's ability to reconstruct

concise and complete attack graphs is invaluable for security analysts, as detection systems cannot guarantee perfect accuracy. Further investigation by security analysts are vitally important for further removing false positives and locating true attacks. Concise and complete attack graphs automatically generated from PIDS can substantially speed up the investigation process, and therefore contribute to swift attack recovery & remediation.

During its tracing on the true attack in the AVIATOR-Sandworm-extended dataset (Figure 9.4), COMMANDER could not identify the attacker's activities on the Workstation_2, the Domain Controller and the Linux Server in the IT domain. The whole network backward tracing from the OT domain, where the initial authentication anomaly was detected, could only lead to the Workstation_1 in the IT domain: In the Sandworm emulation plan, the attacker has gained three initial accesses (edges labeled as 1, 2 and 3, respectively, in Figure 9.4) on different machines during the engagement, instead of an initial access on one machine. Since no system logs from the attacker's machine are present, COMMANDER's logon session-based tracing cannot causally link the three initial accesses from the attacker's machine. Note that COMMANDER cannot detect initial access, nor can any other PIDS. Rather, it adds network sockets with an external IP address in the reconstructed attack graphs as an annotation, suggesting where the initial access could come from. For the sake of simplicity, we only show one such network socket in Figure 9.2 and Figure 9.7. COMMANDER does not perform backward tracing on an edge connecting to network socket, as, in Section 4.3, we show that network connection based-tracing would inevitably lead to the dependency explosion problem.

Moreover, although Linux distributions also implement the concept of logon sessions, popular Linux logging tools like Auditd [85] and CamFlow [218] do not insert logon session ID into system log events, nor do any other Linux logging tools we are aware of. That is, COMMANDER cannot perform logon session-based tracing on Linux system logs yet. We leave it to future work 1) to extend existing Linux logging tools with means for embedding logon session ID into system logs, 2) to design a sound approach for accurately associating initial accesses on different machines from the same attacker.

## 9.6 Summary

In this chapter, we present Commander, a novel PIDS for detecting long-running APT campaigns with the so-called low-and-slow strategy in industrial-sector organizations. Commander's modular design allows it to incorporate specialized detectors for evasion attack techniques, ensuring robust and accurate whole network tracing. In order to evaluate Commander, we extended two MITRE emulation plans, featuring APT groups most known for their attacks against industrial-sector organizations in the past. Our evaluation results demonstrate that Commander can accurately and efficiently detect cross-machine multi-phase APT attacks, and reconstruct concise and insightful attack graphs, crucial for understanding the intrusions and proper attack recovery & remediation. We believe that Commander's ability to perform robust whole network tracing could make it a valuable asset for XDR (eXtended Detection and Response) systems. In future work, we plan to improve existing Linux auditing frameworks by including logon session ID into system logs emitted by them. Moreover, we aim to create a reliable approach for associating initial accesses on different machines in the same network from the same attacker.

# 10 Conclusion and Outlook

This chapter first concludes this thesis by summarizing the proposed systems in Chapter 6 - 9. Then it points out future research directions. Some concrete future work directions are already presented in Chapter 6 - 9, respectively. This chapter complements the outlook, and discusses future directions in a broader context.

## 10.1 Conclusion

In this thesis, we present a novel authentic dataset for research on APT, and three detection and investigation systems for APT attacks, all of which satisfy the five properties of a sound detection system discussed in [21]: reliability, explainability, analytical depth, contextuality, and transferability.

After recognizing several flaws of existing datasets, especially a significant gap between existing datasets' attack scenarios and real-world APT attacks, we create our dataset AVIATOR for stimulating further research on APT detection and investigation. AVIATOR is derived from original and extended MITRE emulation plans, which target realistic enterprise and ICS networks with more sophisticated, cross-machine attacks than those attack scenarios in prior datasets. Due to the absence of criteria for assessing dataset quality, we propose an initial set of quality criteria, through which we compare AVIATOR with prior datasets. AVIATOR outperforms existing datasets in terms of the level of attack scenario complexity & authenticity, dataset operability & usability, and dataset reproducibility & extensibility.

Our findings in terms of existing datasets, and development of AVIATOR as well as criteria for assessing dataset quality, deliver an answer to ***RQ1.1*** - ***RQ1.3*** introduced in Section 1.2. They are discussed in Chapter 6 in detail. Very importantly, the original and extended MITRE emulation plans, and the derived AVIATOR dataset, on the one hand, *enriched* the evaluation our first

APT detection and investigation system CPD, and on the other hand, *enabled* the evaluation of our second and third APT detection and investigation systems: HADES and COMMANDER.

Our first APT detection and investigation system CPD is, to our knowledge, the first specialized cyber persistence detector. The development of CPD is motivated by the high prevalence of persistence techniques among APT actors (answering ***RQ2.1***), and poor performance of prior detection systems owing to a misunderstanding of persistence tactic (answering ***RQ2.2***). By leveraging provenance analytics, CPD significantly reduces false alarms, accurately detects persistence attacks, and outputs succinct attack graphs, enabling multi-phase APT detection and investigation. CPD is based on the insight that effective persistent attacks typically manifest in two phases: the persistence setup and the subsequent persistence execution (answering ***RQ2.3***). By causally coupling these phases, we enhance the ability to detect persistent threats.

First, CPD discerns setups signaling an impending persistent threat and then traces processes linked to remote connections to identify persistence execution activities. A key feature of CPD is the introduction of pseudo-edges, which effectively connect these disjoint phases using data provenance analysis, and expert-guided edges, which empower faster tracing and reduced log size. These edges enable us to detect persistence threats accurately and efficiently. Moreover, we present a new alert triage algorithm that further reduces false positives associated with persistence threats. Evaluations conducted on well-known datasets demonstrate that our system reduces the average false positive rate by 93% compared to state-of-the-art methods. CPD is discussed in Chapter 7 in detail.

Our second system HADES demonstrates, to our knowledge, the first approach for performing accurate and efficient causality-based cross-machine APT detection and investigation. The proposal of HADES's is driven by the fact that cross-machine attack stages are very common in real-world APT attacks (answering ***RQ3.1***), and most prior systems are ill-suited for detecting and investigating cross-machine APT attacks due to their inability to trace across machines (answering ***RQ3.3***). In addition, we find that most cross-machine APT attacks rely on exploiting specific Active Directory functionality (answering ***RQ3.2***). HADES's ability to trace across machines is powered by a novel concept called logon session-based execution partitioning and tracing, which addresses several challenges in cross-machine tracing (answering ***RQ3.4***).

HADES is designed as an efficient on-demand tracing system, which performs cross-machine tracing only when it first identifies an authentication anomaly signifying an ongoing Active Directory-based cross-machine attack, for which we propose a novel lightweight authentication anomaly detection model resulted from our extensive analysis of Active Directory attacks. To triage alerts, we present a new algorithm integrating two key insights we identified in Active Directory attacks. Our evaluation on MITRE's emulation plans shows that HADES outperforms not only popular open-source detection systems but also a prominent commercial AD attack detector. HADES is discussed in Chapter 8 in detail.

Our third system for APT detection and investigation COMMANDER supersedes HADES, and realizes robust cross-machine tracing by eliminating HADES's weakness facing several evasive attack techniques routinely employed by APT actors: persistence, session hijacking, and port forwarding. Whereas port forwarding attacks can directly interrupt HADES's cross-machine tracing, both persistence and session hijacking attacks break accurate intra-machine tracing, on which cross-machine tracing hinges (answering ***RQ4.1***). Hence, we integrate CPD with HADES to prevent accurate cross-machine tracing from being disrupted by persistence attack steps. On top of that, we design another two specialized detectors for port forwarding and session hijacking attack steps, respectively. We present a modular design in COMMANDER, in which the specialized detectors perform preliminary detection individually to deliver critical information for guiding and adjusting the tracing process, hence ensuring robust and correct whole network tracing (answering ***RQ4.2***).

To make provenance-based systems deployable in industrial-sector organizations, we additionally develop parsers for system logs of popular industrial controllers and detection rules with a reference to the MITRE ATT&CK Matrix for ICS. Our evaluations show that COMMANDER accurately detects cross-machine multi-phase APT attacks, outperforms existing detection systems, and delivers succinct and insightful attack graphs for further investigation (answering ***RQ4.3***). Most excitingly, in the attack graphs, COMMANDER accurately attributes the malicious system activities conducted on industrial controllers to the true identity behind those actions, even if the access originates from the enterprise network (answering ***RQ4.4***). COMMANDER is discussed in Chapter 9 in detail.

## 10.2 Outlook

In this thesis, we also identified some constraints of provenance-based security systems that are not addressed in our systems, and should be tackled in future work. First, current provenance-based security systems, including ours, assume that the MITRE ATT&CK Matrix is comprehensive and complete. Future work should investigate the robustness of provenance-based systems in case that attackers use new attack techniques not yet presented in the MITRE ATT&CK Matrix. Second, current provenance-based systems require detailed system logs, leading to log storage problem in the long term. Future work should focus on approaches that can drastically reduce the amount of logs required to be stored or, ideally, to be collected and processed. Third, current provenance-based systems assume the integrity of system logs, future work should introduce methods to actually ensure tamper-proof system logs. Although Microsoft introduced Protected Event Logging [167] for Windows to secure logs from unauthorized access, it is seldom deployed in practice, presumably owing to high overheads. Thus, more efficient solutions for log protection need to be proposed.

# Bibliography


[1] Kaspersky. “What is an advanced persistent threat (APT)?” Accessed: June 2024, [Online]. Available: https://www.kaspersky.com/resource-center/definitions/advanced-persistent-threats.

[2] Cisco. “What is an advanced persistent threat (APT)?” Accessed: June 2024, [Online]. Available: https://www.cisco.com/c/en/us/products/security/advanced-persistent-threat.html.

[3] E. M. Hutchins, M. J. Cloppert, and R. M. Amin. “Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains”. Accessed: June 2024, [Online]. Available: https://www.lockheedmartin.com/content/dam/lockheed-martin/rms/documents/cyber/LM-White-Paper-Intel-Driven-Defense.pdf.

[4] The MITRE Corporation. “Operation dust storm”. Accessed: July 2024, [Online]. Available: https://attack.mitre.org/campaigns/C0016/.

[5] The MITRE Corporation. “Operation sharpshooter”. Accessed: July 2024, [Online]. Available: https://attack.mitre.org/campaigns/C0013/.

[6] The MITRE Corporation. “C0011”. Accessed: July 2024, [Online]. Available: https://attack.mitre.org/campaigns/C0011/.

[7] The MITRE Corporation. “Costaricto”. Accessed: July 2024, [Online]. Available: https://attack.mitre.org/campaigns/C0004/.

[8] The MITRE Corporation. “Operation dream job”. Accessed: July 2024, [Online]. Available: https://attack.mitre.org/campaigns/C0022/.

[9] The MITRE Corporation. “2015 ukraine electric power attack”. Accessed: July 2024, [Online]. Available: https://attack.mitre.org/campaigns/C0028/.

[10] The MITRE Corporation. “2022 ukraine electric power attack”. Accessed: July 2024, [Online]. Available: https://attack.mitre.org/campaigns/C0034/.

[11] The MITRE Corporation. “Unitronics defacement campaign”. Accessed: July 2024, [Online]. Available: https://attack.mitre.org/campaigns/C0031/.

[12] The MITRE Corporation. “Triton safety instrumented system attack”. Accessed: July 2024, [Online]. Available: https://attack.mitre.org/campaigns/C0030/.

[13] The MITRE Corporation. “Maroochy water breach”. Accessed: July 2024, [Online]. Available: https://attack.mitre.org/campaigns/C0020/.

[14] SolarWinds. “What is Advanced Persistent Threat?” Accessed: July 2024, [Online]. Available: https://www.solarwinds.com/resources/it-glossary/advanced-persistent-threat.

[15] O. Moe. “Living off the land binaries, scripts and libraries”. Accessed: May 2023, [Online]. Available: https://lolbas-project.github.io/%5C#.

[16] O. Rumiantseva. “What Are LOLBins?” Accessed: May 2023, [Online]. Available: https://socprime.com/blog/what-are-lolbins/.

[17] Falcon OverWatch Team. “8 LOLBins Every Threat Hunter Should Know”. Accessed: May 2023, [Online]. Available: https://www.crowdstrike.com/blog/8-lolbins-every-threat-hunter-should-know/.

[18] Q. Liu, M. Shoaib, M. U. Rehman, K. Bao, V. Hagenmeyer, and W. U. Hassan, *Accurate and scalable detection and investigation of cyber persistence threats*, 2024. arXiv: 2407.18832 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2407.18832, under review.

[19] Q. Liu, K. Bao, W. U. Hassan, and V. Hagenmeyer, *HADES: Detecting Active Directory attacks via whole network provenance analytics*, 2024. arXiv: 2407.18858 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2407.18858, under review.

[20] The MITRE Corporation. “MITRE Matrix”. Accessed: Jan. 2023, [Online]. Available: https://attack.mitre.org/matrices/enterprise/.

[21] B. A. Alahmadi, L. Axon, and I. Martinovic, “99% false positives: A qualitative study of soc analysts’ perspectives on security alarms”, in *USENIX Security Symposium*, 2022, pp. 2783–2800.

[22] M. N. Hossain, S. M. Milajerdi, J. Wang, B. Eshete, R. Gjomemo, R. Sekar, S. D. Stoller, and V. Venkatakrishnan, “SLEUTH: Real-time attack scenario reconstruction from COTS audit data”, in *USENIX Security Symposium*, 2017, pp. 487–504.

[23] W. U. Hassan, L. Mark, N. Aguse, A. Bates, and T. Moyer, "Towards scalable cluster auditing through grammatical inference over provenance graphs", in *Network and Distributed System Security (NDSS)*, 2018, pp. 1–15.

[24] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates, "NoDoze: Combatting threat alert fatigue with automated provenance triage", in *Network and Distributed System Security (NDSS)*, 2019, pp. 1–15.

[25] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. N. Venkatakrishnan, "HOLMES: Real-time apt detection through correlation of suspicious information flows", in *IEEE Symposium on Security and Privacy (S&P)*, 2019, pp. 1137–1152.

[26] W. U. Hassan, A. Bates, and D. Marino, "Tactical provenance analysis for endpoint detection and response systems", in *IEEE Symposium on Security and Privacy (S&P)*, 2020, pp. 1172–1189.

[27] W. U. Hassan, M. A. Noureddine, P. Datta, and A. Bates, "OmegaLog: High-fidelity attack investigation via transparent multi-layer log analysis", in *Network and Distributed System Security (NDSS)*, 2020, pp. 1–16.

[28] X. Han, T. Pasqueir, A. Bates, J. Mickens, and M. Seltzer, "UNICORN: Runtime provenance-based detector for advanced persistent threats", in *Network and Distributed System Security (NDSS)*, 2020, pp. 1–18.

[29] M. N. Hossain, S. Sheikhi, and R. Sekar, "Combating dependence explosion in forensic analysis using alternative tag propagation semantics", in *IEEE Symposium on Security and Privacy (S&P)*, 2020, pp. 1139–1155.

[30] C. Xiong, T. Zhu, W. Dong, L. Ruan, R. Yang, Y. Cheng, Y. Chen, S. Cheng, and X. Chen, "Conan: A practical real-time apt detection system with high accuracy and efficiency", *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 551–565, 2022. DOI: 10.1109/TDSC.2020.2971484.

[31] H. Irshad, G. Ciocarlie, A. Gehani, V. Yegneswaran, K. H. Lee, J. Patel, S. Jha, Y. Kwon, D. Xu, and X. Zhang, "Trace: Enterprise-wide provenance tracking for real-time apt detection", *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4363–4376, 2021.

[32] A. Alsaheel, Y. Nan, S. Ma, L. Yu, G. Walkup, Z. B. Celik, X. Zhang, and D. Xu, "Atlas: A sequence-based learning approach for attack investigation", in *USENIX Security Symposium*, 2021, pp. 3005–3022.

[33] J. Zeng, X. Wang, J. Liu, Y. Chen, Z. Liang, T.-S. Chua, and Z. L. Chua, "Shadewatcher: Recommendation-guided cyber threat analysis using system audit records", in *IEEE Symposium on Security and Privacy (S&P)*, 2022, pp. 489–506.

[34] F. Dong, L. Wang, X. Nie, F. Shao, H. Wang, D. Li, X. Luo, and X. Xiao, "DISTDET: A Cost-Effective distributed cyber threat detection system", in *USENIX Security Symposium*, 2023, pp. 6575–6592.

[35] Z. Cheng, Q. Lv, J. Liang, Y. Wang, D. Sun, T. Pasquier, and X. Han, "KAIROS: Practical intrusion detection and investigation using whole-system provenance", in *IEEE Symposium on Security and Privacy (S&P)*, 2024, pp. 9–28.

[36] M. U. Rehman, H. Ahmadi, and W. U. Hassan, "Flash: A comprehensive approach to intrusion detection via provenance graph representation learning", in *IEEE Symposium on Security and Privacy (S&P)*, 2024, pp. 142–161.

[37] CrowdStrike, Inc., "CrowdStrike 2023 global threat report", 2023. [Online]. Available: https://www.crowdstrike.com/global-threat-report/.

[38] CrowdStrike, Inc., "CrowdStrike 2023 threat hunting report", 2023. [Online]. Available: https://www.crowdstrike.com/resources/reports/threat-hunting-report/.

[39] V. Shastri. "Attackers Set Sights on Active Directory: Understanding Your Identity Exposure". Accessed: Dec. 2023, [Online]. Available: https://www.crowdstrike.com/blog/attackers-set-sights-on-active-directory-understanding-your-identity-exposure/.

[40] V. Shastri. "Endpoint and Identity Security: A Critical Combination to Stop Modern Attacks". Accessed: Dec. 2023, [Online]. Available: https://www.crowdstrike.com/blog/unifying-endpoint-and-identity-security/.

[41] Dragos. "Credentials across it and ot environments". Accessed: July 2024, [Online]. Available: https://www.dragos.com/blog/minimizing-the-consequences-of-shared-credentials-across-it-and-ot-environments/.

[42] M. Crockett. "Why 86% of organizations are increasing their investment in active directory security". Accessed: Dec. 2023, [Online]. Available: https://securityboulevard.com/2021/11/why-86-of-organizations-are-increasing-their-investment-in-active-directory-security/.

[43] Q. Liu, V. Hagenmeyer, and H. B. Keller, "A review of rule learning-based intrusion detection systems and their prospects in smart grids", *IEEE Access*, vol. 9, pp. 57 542–57 564, 2021. DOI: 10.1109/ACCESS.2021.3071263.

[44] Q. Liu, H. B. Keller, and V. Hagenmeyer, "A bayesian rule learning based intrusion detection system for the MQTT communication protocol", in *Proceedings of the 16th International Conference on Availability, Reliability and Security*, 2021. DOI: 10.1145/3465481.3470046.

[45] S. Krishnamoorthi and J. Carleton, "Active directory holds the keys to your kingdom, but is it secure?", 2020. [Online]. Available: https://www.frost.com/frost-perspectives/active-directory-holds-the-keys-to-your-kingdom-but-is-it-secure/.

[46] The MITRE Corporation. "MITRE Adversary Emulation Library". Accessed: Jan. 2023, [Online]. Available: https://github.com/center-for-threat-informed-defense/adversary_emulation_library.

[47] Q. Liu, K. Bao, and V. Hagenmeyer, "COMMANDER: A robust cross-machine multi-phase Advanced Persistent Threat detector", under review.

[48] The MITRE Corporation. "MITRE ATT&CK". Accessed: Jan. 2023, [Online]. Available: https://attack.mitre.org.

[49] Siemens. "Digital protection relays and control - SIPROTEC 5". Accessed: June 2024, [Online]. Available: https://www.siemens.com/global/en/products/energy/energy-automation-and-smart-grid/protection-relays-and-control/siprotec-5.html.

[50] Hitachi Energy. "Rtu500 series function and software". Accessed: June 2024, [Online]. Available: https://www.hitachienergy.com/products-and-solutions/substation-automation-protection-and-control/products/remote-terminal-units/rtu500-series-function-and-software.

[51] The MITRE Corporation. "ICS Matrix". Accessed: June 2024, [Online]. Available: https://attack.mitre.org/matrices/ics/.

[52] Mandiant, "Mandiant M-Trends 2023", 2023. [Online]. Available: https://www.mandiant.com/resources/reports.

[53] Microsoft Threat Intelligence. "Deep dive into the Solorigate second-stage activation". Accessed: Oct. 2023, [Online]. Available: https://www.microsoft.com/en-us/security/blog/2021/01/20/deep-dive-into-the-solorigate-second-stage-activation-from-sunburst-to-teardrop-and-raindrop/.

[54] J. Slowik. "Anatomy of an attack: Detecting and defeating crashoverride". Accessed: June 2024, [Online]. Available: https://www.dragos.com/wp-content/uploads/CRASHOVERRIDE2018.pdf.

[55] Trellix, "Trellix Threat Report 2023", 2023. [Online]. Available: https://www.trellix.com/advanced-research-center/threat-reports/feb-2023/.

[56] S. Andersen and V. Abella. "Changes to functionality in microsoft windows xp service pack 2, part 3: Memory protection technologies, data execution prevention". Accessed: July 2024, [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-xp/bb457155(v=technet.10).

[57] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti, "Control-flow integrity", in *ACM Conference on Computer and Communications Security (CCS)*, Association for Computing Machinery, 2005, pp. 340–353.

[58] PAX Team. "PAX Address Space Layout Randomization (ASLR)". Accessed: July 2024, [Online]. Available: https://pax.grsecurity.net/docs/aslr.txt.

[59] Q. Liu, K. Bao, and V. Hagenmeyer, "Binary exploitation in industrial control systems: Past, present and future", *IEEE Access*, vol. 10, pp. 48 242–48 273, 2022. DOI: 10.1109/ACCESS.2022.3171922.

[60] Q. Liu, K. Bao, and V. Hagenmeyer, "AVIATOR: A MITRE emulation plan-derived living dataset for Advanced Persistent Threat detection and investigation", in *Proceedings of 2024 IEEE International Conference on Big Data*, 2024, in print.

[61] L. Szekeres, M. Payer, T. Wei, and D. Song, "Sok: Eternal war in memory", in *IEEE Symposium on Security and Privacy (S&P)*, 2013, pp. 48–62.

[62] The MITRE Corporation. "Initial access". Accessed: July 2024, [Online]. Available: https://attack.mitre.org/tactics/TA0001/.

[63] F. Barr-Smith, X. Ugarte-Pedrero, M. Graziano, R. Spolaor, and I. Martinovic, "Survivalism: Systematic analysis of windows malware living-off-the-land", in *IEEE Symposium on Security and Privacy (S&P)*, 2021, pp. 1557–1574. DOI: 10.1109/SP40001.2021.00047.

[64] T. Ongun, J. W. Stokes, J. B. Or, K. Tian, F. Tajaddodianfar, J. Neil, C. Seifert, A. Oprea, and J. C. Platt, "Living-off-the-land command detection using active learning", in *Proceedings of the 24th International Symposium on Research in Attacks, Intrusions and Defenses*, ser. RAID '21, 2021, pp. 442–455. DOI: 10.1145/3471621.3471858.

[65] *Nmap*, https://nmap.org/, Last accessed: May, 2024.

[66] The MITRE Corporation. "Mobile Matrix". Accessed: June 2024, [Online]. Available: https://attack.mitre.org/tactics/mobile/.

[67] V. Kuznetsov, L. Szekeres, M. Payer, G. Candea, R. Sekar, and D. Song, "Code-Pointer integrity", in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, USENIX Association, 2014, pp. 147–163.

[68] DARPA. "Transparent computing". Accessed: June 2024, [Online]. Available: https://www.darpa.mil/program/transparent-computing.

[69] A. D. Keromytis. "DARPA Transparent Computing E3". Accessed: Sept. 2023, [Online]. Available: https://github.com/darpa-i2o/Transparent-Computing/blob/master/README-E3.md.

[70] J. Torrey. "DARPA Transparent Computing". Accessed: Sept. 2023, [Online]. Available: https://github.com/darpa-i2o/Transparent-Computing.

[71] M. van Opstal and W. Arbaugh. "DARPA OpTC". Accessed: Sept. 2023, [Online]. Available: https://github.com/FiveDirections/OpTC-data.

[72] Broadcom. "Symantec Endpoint Security (SES)". Accessed: July 2024, [Online]. Available: https://techdocs.broadcom.com/us/en/symantec-security-software/endpoint-security-and-management/endpoint-security/sescloud/Endpoint-Detection-and-Response.html.

[73] C. Trynoga. "McAfee Defenders Blog: Reality Check for your Defenses". Accessed: July 2024, [Online]. Available: https://www.mcafee.com/blogs/other-blogs/mcafee-labs/mcafee-defenders-blog-reality-check-for-your-defenses/.

[74] SigmaHQ. "Sigma". Accessed: Sept. 2023, [Online]. Available: https://github.com/SigmaHQ/sigma.

[75] Google Security Operations. “Chronicle Detection Rules”. Accessed: Sept. 2023, [Online]. Available: https://github.com/chronicle/detection-rules.

[76] E. Segal. “Alert Fatigue”. Accessed: Aug. 2023, [Online]. Available: https://www.forbes.com/sites/edwardsegal/2021/11/08/alert-fatigue-can-lead-to-missed-cyber-threats-and-staff-retentionrecruitment-issues-study/?sh=4c96871035c9.

[77] C. Robinson, “In Cybersecurity Every Alert Matters”, 2021. [Online]. Available: https://www.criticalstart.com/wp-content/uploads/2021/11/US48277521_TLWP.pdf.

[78] M. Wojtasiak, “The defenders’ dilemma”, 2023. [Online]. Available: https://info.vectra.ai/state-of-threat-detection.

[79] CRITICALSTART, “The Impact of Security Alert Overload”, 2019. [Online]. Available: https://www.criticalstart.com/wp-content/uploads/2021/02/CS_Report-The-Impact-of-Security-Alert-Overload.pdf.

[80] S. Ma, X. Zhang, and D. Xu, “ProTracer: Towards practical provenance tracing by alternating between logging and tainting”, in *Network and Distributed System Security (NDSS)*, 2016, pp. 1–15.

[81] F. Yang, J. Xu, C. Xiong, Z. Li, and K. Zhang, “Prographer: An anomaly detection system based on provenance graph embedding”, in *USENIX Security Symposium*, 2023, pp. 4355–4372.

[82] F. Dong, S. Li, P. Jiang, D. Li, H. Wang, L. Huang, X. Xiao, J. Chen, X. Luo, Y. Guo, and X. Chen, “Are we there yet? an industrial viewpoint on provenance-based endpoint detection and response tools”, in *ACM Conference on Computer and Communications Security (CCS)*, 2023, pp. 2396–2410.

[83] D. Marshall. “Event Tracing for Windows (ETW)”. Accessed: Dec. 2023, [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/devtest/event-tracing-for-windows--etw-.

[84] M. Russinovich and T. Garnier. “System Monitor”. Accessed: Feb. 2023, [Online]. Available: https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon.

[85] S. Grubb, *The Linux audit daemon*, Accessed: Feb. 2023. [Online]. Available: https://linux.die.net/man/8/auditd.

[86] The MITRE Corporation. “APT29 emulation plan”. Accessed: Oct. 2023, [Online]. Available: https://github.com/center-for-threat-informed-defense/adversary_emulation_library/tree/master/apt29.

[87] The MITRE Corporation. “Sandworm emulation plan”. Accessed: Feb. 2023, [Online]. Available: https://github.com/center-for-threat-informed-defense/adversary_emulation_library/tree/master/sandworm.

[88] The MITRE Corporation. “WizardSpider emulation plan”. Accessed: Oct. 2023, [Online]. Available: https://github.com/center-for-threat-informed-defense/adversary_emulation_library/tree/master/wizard_spider.

[89] The MITRE Corporation. “Oilrig emulation plan”. Accessed: Oct. 2023, [Online]. Available: https://github.com/center-for-threat-informed-defense/adversary_emulation_library/tree/master/oilrig.

[90] G. Ho, M. Dhiman, D. Akhawe, V. Paxson, S. Savage, G. M. Voelker, and D. A. Wagner, “Hopper: Modeling and detecting lateral movement”, in *USENIX Security Symposium*, 2021, pp. 3093–3110.

[91] I. J. King and H. H. Huang, “Euler: Detecting network lateral movement via scalable temporal link prediction”, in *Network and Distributed System Security (NDSS)*, 2022, pp. 1–16.

[92] G. Ho, A. Sharma, M. Javed, V. Paxson, and D. Wagner, “Detecting credential spearphishing in enterprise settings”, in *USENIX Security Symposium*, 2017, pp. 469–485.

[93] C. Novo and R. Morla, “Flow-based detection and proxy-based evasion of encrypted malware c2 traffic”, in *Proc. ACM Workshop on Artificial Intelligence and Security*, 2020, pp. 83–91.

[94] A. Alageel and S. Maffeis, “Hawk-eye: Holistic detection of apt command and control domains”, in *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, ser. SAC ’21, 2021, pp. 1664–1673. DOI: 10.1145/3412841.3442040.

[95] Y. Ozery, A. Nadler, and A. Shabtai, “Information based heavy hitters for real-time dns data exfiltration detection”, in *Network and Distributed System Security (NDSS)*, 2024, pp. 1–15.

[96] A. Oprea, Z. Li, T.-F. Yen, S. H. Chin, and S. Alrwais, “Detection of early-stage enterprise infection by mining large-scale log data”, in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015, pp. 45–56. DOI: 10.1109/DSN.2015.14.

[97] F. Wilkens, F. Ortmann, S. Haas, M. Vallentin, and M. Fischer, "Multi-stage attack detection via kill chain state machines", in *Proceedings of the 3rd Workshop on Cyber-Security Arms Race*, ser. CYSARM '21, 2021, pp. 13–24. DOI: 10.1145/3474374.3486918.

[98] The Zeek Project. "About zeek". Accessed: March 2024, [Online]. Available: https://docs.zeek.org/en/master/about.html.

[99] S. Haas and M. 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*, ser. SAC '18, 2018, pp. 979–988. DOI: 10.1145/3167132.3167239.

[100] P. Ah-Fat, M. Huth, R. Mead, T. Burrell, and J. Neil, "Effective detection of credential thefts from windows memory: Learning access behaviours to local security authority subsystem service", in *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, San Sebastian, 2020, pp. 181–194.

[101] W. Han, J. Xue, Y. Wang, F. Zhang, and X. Gao, "APTMalInsight: Identify and cognize apt malware based on system call information and ontology knowledge framework", *Information Sciences*, vol. 546, pp. 633–664, 2021. DOI: https://doi.org/10.1016/j.ins.2020.08.095. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025520308628.

[102] G. Shenderovitz and N. Nissim, "Bon-apt: Detection, attribution, and explainability of apt malware using temporal segmentation of api calls", *Computers & Security*, vol. 142, p. 103 862, 2024. DOI: https://doi.org/10.1016/j.cose.2024.103862. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404824001639.

[103] Z. Xu, Z. Wu, Z. Li, K. Jee, J. Rhee, X. Xiao, F. Xu, H. Wang, and G. Jiang, "High fidelity data reduction for big data security dependency analyses", in *ACM Conference on Computer and Communications Security (CCS)*, 2016, pp. 504–516.

[104] Y. Tang, D. Li, Z. Li, M. Zhang, K. Jee, X. Xiao, Z. Wu, J. Rhee, F. Xu, and Q. Li, "Nodemerge: Template based efficient data reduction for big-data causality analysis", in *ACM Conference on Computer and Communications Security (CCS)*, 2018, pp. 1324–1337.

[105] M. N. Hossain, J. Wang, R. Sekar, and S. D. Stoller, "Dependence-Preserving data compaction for scalable forensic analysis", in *USENIX Security Symposium*, 2018, pp. 1723–1740.

[106] N. Michael, J. Mink, J. Liu, S. Gaur, W. U. Hassan, and A. Bates, "On the forensic validity of approximated audit logs", in *Proceedings of the 36th Annual Computer Security Applications Conference*, ser. ACSAC '20, 2020, pp. 189–202. DOI: 10.1145/3427228.3427272.

[107] P. Fei, Z. Li, Z. Wang, X. Yu, D. Li, and K. Jee, "SEAL: Storage-efficient causality analysis on enterprise logs with query-friendly compression", in *30th USENIX Security Symposium (USENIX Security 21)*, USENIX Association, 2021, pp. 2987–3004. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/fei.

[108] H. Ding, S. Yan, J. Zhai, and S. Ma, "Elise: A storage efficient logging system powered by redundancy reduction and representation learning", in *USENIX Security Symposium*, 2021, pp. 3023–3040.

[109] M. A. Inam, Y. Chen, A. Goyal, J. Liu, J. Mink, N. Michael, S. Gaur, A. Bates, and W. U. Hassan, "Sok: History is a vast early warning system: Auditing the provenance of system intrusions", in *IEEE Symposium on Security and Privacy (S&P)*, 2023, pp. 2620–2638.

[110] K. H. Lee, X. Zhang, and D. Xu, "High accuracy attack provenance via binary-based execution partition", in *Network and Distributed System Security (NDSS)*, 2013, pp. 1–16.

[111] S. Ma, K. H. Lee, C. H. Kim, J. Rhee, X. Zhang, and D. Xu, "Accurate, low cost and instrumentation-free security audit logging for Windows", in *Annual Computer Security Applications Conference (ACSAC)*, 2015, pp. 401–410.

[112] S. Ma, J. Zhai, F. Wang, K. H. Lee, X. Zhang, and D. Xu, "Mpi: Multiple perspective attack investigation with semantic aware execution partitioning", in *USENIX Security Symposium*, 2017, pp. 1111–1128.

[113] F. Welter, F. Wilkens, and M. Fischer, "Tell me more: Black box explainability for apt detection on system provenance graphs", in *ICC 2023 - IEEE International Conference on Communications*, 2023, pp. 3817–3823. DOI: 10.1109/ICC45041.2023.10279468.

[114] S. Wang, Z. Wang, T. Zhou, H. Sun, X. Yin, D. Han, H. Zhang, X. Shi, and J. Yang, "Threatrace: Detecting and tracing host-based threats in node level through provenance graph learning", *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 3972–3987, 2022.

[115] Z. Jia, Y. Xiong, Y. Nan, Y. Zhang, J. Zhao, and M. Wen, *Magic: Detecting advanced persistent threats via masked graph representation learning*, 2023. arXiv: 2310.09831 [cs.CR].

[116] E. Manzoor, S. M. Milajerdi, and L. Akoglu, "Fast memory-efficient anomaly detection in streaming heterogeneous graphs", ser. KDD '16, 2016, pp. 1035–1044. DOI: 10.1145/2939672.2939783.

[117] T. Chen, C. Dong, M. Lv, Q. Song, H. Liu, T. Zhu, K. Xu, L. Chen, S. Ji, and Y. Fan, "Apt-kgl: An intelligent apt detection system based on threat knowledge and heterogeneous provenance graph learning", *IEEE Transactions on Dependable and Secure Computing*, pp. 1–15, 2022. DOI: 10.1109/TDSC.2022.3229472.

[118] J. Zeng, Z. L. Chua, Y. Chen, K. Ji, Z. Liang, and J. Mao, "Watson: Abstracting behaviors from audit logs via aggregation of contextual semantics", in *Network and Distributed System Security (NDSS)*, 2021, pp. 1–18.

[119] P. Fang, P. Gao, C. Liu, E. Ayday, K. Jee, T. Wang, Y. ( Ye, Z. Liu, and X. Xiao, "Back-Propagating system dependency impact for attack investigation", in *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA, 2022, pp. 2461–2478.

[120] The MITRE Corporation. "MITRE T1543.003". Accessed: Feb. 2023, [Online]. Available: https://attack.mitre.org/techniques/T1543/003/.

[121] The MITRE Corporation. "MITRE T1543.002". Accessed: Feb. 2023, [Online]. Available: https://attack.mitre.org/techniques/T1543/002/.

[122] S. Myneni, A. Chowdhary, A. Sabur, S. Sengupta, G. Agrawal, D. Huang, and M. Kang, "Dapt 2020 - constructing a benchmark dataset for advanced persistent threats", in *Deployable Machine Learning for Security Defense*, G. Wang, A. Ciptadi, and A. Ahmadzadeh, Eds., Cham: Springer International Publishing, 2020, pp. 138–163.

[123] S. Myneni, K. Jha, A. Sabur, G. Agrawal, Y. Deng, A. Chowdhary, and D. Huang, "Unraveled — a semi-synthetic dataset for advanced persistent threats", *Computer Networks*, vol. 227, p. 109 688, 2023. DOI: https://doi.org/10.1016/j.comnet.2023.109688. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128623001330.

[124] A. Riddle, K. Westfall, and A. Bates, *Atlasv2: Atlas attack engagements, version 2*, 2023. arXiv: 2401.01341 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2401.01341.

[125] S. S. Karim, M. Afzal, W. Iqbal, and D. A. Abri, "Advanced persistent threat (apt) and intrusion detection evaluation dataset for linux systems 2024", *Data in Brief*, vol. 54, p. 110 290, 2024. DOI: https://doi.org/10.1016/j.dib.2024.110290. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352340924002592.

[126] P. Paganini. "SolarWinds hack: the mystery of one of the biggest cyberattacks ever". Accessed: Oct. 2023, [Online]. Available: https://cybernews.com/security/solarwinds-hack-the-mystery-of-one-of-the-biggest-cyberattacks-ever/.

[127] CrowdStrike Intelligence Team. "SUNSPOT: An Implant in the Build Process". Accessed: Oct. 2023, [Online]. Available: https://www.crowdstrike.com/blog/sunspot-malware-technical-analysis/.

[128] The MITRE Corporation. "Sandworm Team". Accessed: June 2023, [Online]. Available: https://attack.mitre.org/groups/G0034/.

[129] French Cybersecurity Agency, "Sandworm intrusion set campaign targeting Centreon systems", 2021. [Online]. Available: https://www.cert.ssi.gouv.fr/uploads/CERTFR-2021-CTI-005.pdf.

[130] Pulsedive. "P.A.S. Webshell". Accessed: Sept. 2023, [Online]. Available: https://pulsedive.com/threat/P.A.S.%5C%20Webshell.

[131] The MITRE Corporation. "MITRE Attack Stix Data". Accessed: April 2023, [Online]. Available: https://github.com/mitre-attack/attack-stix-data.

[132] The MITRE Corporation. "MITRE T1547.001". Accessed: May 2023, [Online]. Available: https://attack.mitre.org/techniques/T1547/001/.

[133] The MITRE Corporation. "T1078". Accessed: June 2023, [Online]. Available: https://attack.mitre.org/techniques/T1078/.

[134] The MITRE Corporation. "T1098004". Accessed: June 2023, [Online]. Available: https://attack.mitre.org/techniques/T1098/004/.

[135] The MITRE Corporation. "T1053". Accessed: June 2023, [Online]. Available: https://attack.mitre.org/techniques/T1053/.

[136] The MITRE Corporation. "T1546.004". Accessed: June 2024, [Online]. Available: https://attack.mitre.org/techniques/T1546/004/.

[137] Q. Wang, W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, J. Rhee, Z. Chen, W. Cheng, C. A. Gunter, *et al.*, "You are what you do: Hunting stealthy malware via data provenance analysis", in *Network and Distributed System Security (NDSS)*, 2020, pp. 1–17.

[138] The MITRE Corporation. “MITRE T1558.003”. Accessed: Dec. 2023, [Online]. Available: https://attack.mitre.org/techniques/T1558/003/.

[139] The MITRE Corporation. “MITRE T1550.002”. Accessed: Dec. 2023, [Online]. Available: https://attack.mitre.org/techniques/T1550/002/.

[140] A. Talyanski. “Nopac exploit: Latest microsoft AD flaw may lead to total domain compromise in seconds”. Accessed: June 2024, [Online]. Available: https://www.crowdstrike.com/blog/nopac-exploit-latest-microsoft-ad-flaw-may-lead-to-total-domain-compromise/.

[141] Tenable, Inc. “A gloabl threat to enterprises: The impact of Active Directory attacks”. Accessed: June 2024, [Online]. Available: https://de.tenable.com/whitepapers/a-global-threat-to-enterprises-the-impact-of-ad-attacks?page=2.

[142] The MITRE Corporation. “Persistence”. Accessed: June 2024, [Online]. Available: https://attack.mitre.org/tactics/TA0003/.

[143] The MITRE Corporation. “T1563”. Accessed: June 2024, [Online]. Available: https://attack.mitre.org/techniques/T1563/.

[144] The MITRE Corporation. “T1090.001 ”. Accessed: June 2024, [Online]. Available: https://attack.mitre.org/techniques/T1090/001/.

[145] I. Koecher. “Are disconnected rdp sessions ticking time bombs in your network?” Accessed: June 2024, [Online]. Available: https://www.eventsentry.com/blog/2022/04/are-disconnected-rdp-sessions-ticking-time-bombs-in-your-network.html.

[146] Elastic. “Elastic Detection Rules”. Accessed: Sept. 2023, [Online]. Available: https://github.com/elastic/detection-rules.

[147] The MITRE Corporation. “MITRE Engenuity”. Accessed: Jan. 2023, [Online]. Available: https://attackevals.mitre-engenuity.org/.

[148] Defense Science Board Task Force. “The role and status of DoD red teaming activities”. Accessed: June 2024, [Online]. Available: https://web.archive.org/web/20090419081417/http://www.acq.osd.mil/dsb/reports/redteam.pdf.

[149] Booz Allen Hamilton. “When the lights went out”. Accessed: June 2024, [Online]. Available: https://www.boozallen.com/content/dam/boozallen/documents/2016/09/ukraine-report-when-the-lights-went-out.pdf.

[150] A. Cherepanov. “Win32/industroyer: A new threat for industrial control systems”. Accessed: June 2024, [Online]. Available: https://web-assets.esetstatic.com/wls/2017/06/Win32_Industroyer.pdf.

[151] Mandiant. “Sandworm disrupts power in ukraine using a novel attack against operational technology”. Accessed: June 2024, [Online]. Available: https://cloud.google.com/blog/topics/threat-intelligence/sandworm-disrupts-power-ukraine-operational-technology/.

[152] The MITRE Corporation. “OilRig”. Accessed: June 2024, [Online]. Available: https://attack.mitre.org/groups/G0049/.

[153] Mandiant. “New targeted attack in the middle east by APT34”. Accessed: June 2024, [Online]. Available: https://cloud.google.com/blog/topics/threat-intelligence/targeted-attack-in-middle-east-by-apt34/.

[154] Dragos Threat Intelligence. “CHRYSENE threat group operations”. Accessed: June 2024, [Online]. Available: https://www.dragos.com/threat/chrysene/.

[155] Microsoft. “Security auditing”. Accessed: May 2023, [Online]. Available: https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-10/security/threat-protection/auditing/security-auditing-overview.

[156] Wireshark Foundation. “Wireshark”. Accessed: June 2024, [Online]. Available: https://www.wireshark.org/.

[157] The MITRE Corporation. “Turla”. Accessed: June 2024, [Online]. Available: https://github.com/center-for-threat-informed-defense/adversary_emulation_library/tree/master/turla.

[158] The MITRE Corporation. “OceanLotus emulation plan”. Accessed: June 2024, [Online]. Available: https://github.com/center-for-threat-informed-defense/adversary_emulation_library/tree/master/ocean_lotus.

[159] The MITRE Corporation. “Blind Eagle emulation plan”. Accessed: June 2024, [Online]. Available: https://github.com/center-for-threat-informed-defense/adversary_emulation_library/tree/master/blind_eagle.

[160] NIST. “Guide to industrial control systems (ICS) security”. Accessed: July 2024, [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r2.pdf.

[161] S. Mathezer. “Introduction to ics security part 3”. Accessed: July 2024, [Online]. Available: https://www.sans.org/blog/introduction-to-ics-security-part-3/.

[162] IBM Research. “What is offensive security?” Accessed: June 2024, [Online]. Available: https://www.ibm.com/topics/offensive-security.

[163] D. Marshall. “NT Kernel Logger”. Accessed: Feb. 2023, [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/devtest/nt-kernel-logger-trace-session.

[164] Microsoft. “CreateProcessA function”. Accessed: October 2024, [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessa.

[165] Microsoft. “PsSetCreateProcessNotifyRoutine function”. Accessed: October 2024, [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/ntddk/nf-ntddk-pssetcreateprocessnotifyroutine.

[166] S. Wheeler. “Cmdlet overview”. Accessed: June 2023, [Online]. Available: https://learn.microsoft.com/de-de/powershell/scripting/developer/cmdlet/cmdlet-overview?view=powershell-7.4.

[167] S. Wheeler and M. Lombardi. “About Logging Windows”. Accessed: April 2023, [Online]. Available: https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_logging_windows?view=powershell-7.3.

[168] Red Hat, Inc. “Red Hat Security Guide”. Accessed: Feb. 2023, [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/security_guide/chap-system_auditing.

[169] Elastic NV. “Elasticsearch”. Accessed: Sept. 2023, [Online]. Available: https://www.elastic.co/.

[170] Elastic NV. “Logstash”. Accessed: June 2024, [Online]. Available: https://www.elastic.co/logstash.

[171] Elastic NV. “Kibana”. Accessed: June 2023, [Online]. Available: https://www.elastic.co/kibana.

[172] Elastic NV. “Lightweight shipper for audit data”. Accessed: June 2023, [Online]. Available: https://www.elastic.co/beats/auditbeat/.

[173] Elastic NV. “Filebeat”. Accessed: June 2024, [Online]. Available: https://www.elastic.co/beats/filebeat.

[174] Elastic NV. “Lightweight shipper for windows event logs”. Accessed: June 2023, [Online]. Available: https://www.elastic.co/beats/winlogbeat/.

[175] Elastic NV. “EQL search”. Accessed: Sept. 2023, [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/eql.html.

[176] NetworkX developers. “NetworkX”. Accessed: Sept. 2023, [Online]. Available: https://networkx.org/.

[177] West Health Institute. “PyVis”. Accessed: Sept. 2023, [Online]. Available: https://pyvis.readthedocs.io/en/latest/.

[178] Wazuh, Inc. “The open source security platform”. Accessed: July 2024, [Online]. Available: https://wazuh.com/.

[179] Red Canary. “Atomic Red Team”. Accessed: Jan. 2023, [Online]. Available: https://atomicredteam.io/.

[180] R. Uetz, M. Herzog, L. Hackländer, S. Schwarz, and M. Henze, *You cannot escape me: Detecting evasions of SIEM rules in enterprise networks*, 2023. arXiv: 2311.10197 [cs.CR].

[181] M. Russinovich. “Process Monitor”. Accessed: Feb. 2023, [Online]. Available: https://learn.microsoft.com/en-us/sysinternals/downloads/procmon.

[182] A. Allievi, A. Ionescu, D. A. Solomon, K. Chase, and M. E. Russinovich, *Windows Internals, Part 2, 7th Edition*. Microsoft Press, 2022.

[183] The MITRE Corporation. “MITRE T1053.005”. Accessed: Feb. 2023, [Online]. Available: https://attack.mitre.org/techniques/T1053/005/.

[184] The MITRE Corporation. “T1098”. Accessed: June 2023, [Online]. Available: https://attack.mitre.org/techniques/T1098/.

[185] The MITRE Corporation. “T1136”. Accessed: June 2023, [Online]. Available: https://attack.mitre.org/techniques/T1136/.

[186] The MITRE Corporation. “APT29”. Accessed: April 2023, [Online]. Available: https://attack.mitre.org/groups/G0016/.

[187] The MITRE Corporation. “Wizard Spider”. Accessed: April 2023, [Online]. Available: https://attack.mitre.org/groups/G0102/.

[188] The MITRE Corporation. “T1003”. Accessed: June 2023, [Online]. Available: https://attack.mitre.org/techniques/T1003/.

[189] S. White. “Windows Remote Management”. Accessed: March 2023, [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/winrm/portal.

[190] The MITRE Corporation. “Carbanak”. Accessed: April 2023, [Online]. Available: https://attack.mitre.org/groups/G0008/.

[191] The MITRE Corporation. “T1018”. Accessed: June 2023, [Online]. Available: https://attack.mitre.org/techniques/T1018/.

[192] The MITRE Corporation. “T1105”. Accessed: June 2023, [Online]. Available: https://attack.mitre.org/techniques/T1105/.

[193] A. Riddle, K. Westfall, and A. Bates. “ATLASv2”. Accessed: Oct. 2023, [Online]. Available: https://bitbucket.org/sts-lab/atlasv2/src/master/.

[194] VMware LLC. “Carbon Black Cloud”. Accessed: Oct. 2023, [Online]. Available: https://www.vmware.com/products/carbon-black-cloud.html.

[195] Elastic. “A elastic rule sample”. Accessed: Sept. 2023, [Online]. Available: https://github.com/elastic/detection-rules/blob/main/rules/windows/persistence_registry_uncommon.toml.

[196] SigmaHQ. “A sigma rule sample”. Accessed: Sept. 2023, [Online]. Available: https://github.com/SigmaHQ/sigma/blob/master/rules/windows/file/file_event/file_event_win_creation_unquoted_service_path.yml.

[197] Google Security Operations. “A chronicle rule sample”. Accessed: Sept. 2023, [Online]. Available: https://github.com/chronicle/detection-rules/blob/main/mitre_attack/T1053_005_windows_creation_of_scheduled_task.yaral.

[198] The MITRE Corporation. “T1574.001”. Accessed: June 2023, [Online]. Available: https://attack.mitre.org/techniques/T1574/001/.

[199] The MITRE Corporation. “T1574.002”. Accessed: June 2023, [Online]. Available: https://attack.mitre.org/techniques/T1574/002/.

[200] F. Pedregosa and P. Gervais. “Memory profiler”. Accessed: Oct. 2023, [Online]. Available: https://pypi.org/project/memory-profiler/.

[201] The MITRE Corporation. “T1546.003”. Accessed: June 2024, [Online]. Available: https://attack.mitre.org/techniques/T1546/003/.

[202] The MITRE Corporation. “Golden Ticket”. Accessed: Jan. 2024, [Online]. Available: https://attack.mitre.org/techniques/T1558/001/.

[203] Microsoft. “LSA logon sessions”. Accessed: Feb. 2024, [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/secauthn/lsa-logon-sessions.

[204] M. E. Russinovich and A. Margosis, *Troubleshooting with the Windows Sysinternals Tools, 2nd Edition*. Microsoft Press, 2016.

[205] A. Miroshnikov, *Windows Security Monitoring: Scenarios and Patterns*. Wiley, 2018.

[206] J. Forshaw, *Windows Security Internals: A Deep Dive into Windows Authentication, Authorization, and Auditing*. No Starch Press, 2024.

[207] The MITRE Corporation. “T1021.001”. Accessed: Jan. 2024, [Online]. Available: https://attack.mitre.org/techniques/T1021/001/.

[208] Microsoft. “Fast User Switching”. Accessed: Feb. 2024, [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/shell/fast-user-switching.

[209] Microsoft. “User Account Control”. Accessed: Feb. 2024, [Online]. Available: https://learn.microsoft.com/en-us/windows/security/application-security/application-control/user-account-control/.

[210] The MITRE Corporation. “MITRE ATT&CK Campaigns”. Accessed: May 2024, [Online]. Available: https://attack.mitre.org/campaigns/.

[211] X. Bouwman, H. Griffioen, J. Egbers, C. Doerr, B. Klievink, and M. van Eeten, “A different cup of TI? the added value of commercial threat intelligence”, in *USENIX Security Symposium*, 2020, pp. 433–450.

[212] E. Mirolyubov, M. Taggett, F. Hinner, and N. Patel, “Magic quadrant for endpoint protection platforms”, 2023. [Online]. Available: https://www.gartner.com/doc/reprints?id=1-2FFCXFOM&ct=231025&st=sb.

[213] A. Mellen, “The Forrester New Wave: Extended Detection And Response (XDR) Providers, Q4 2021”, 2021. [Online]. Available: https://www.forrester.com/report/the-forrester-new-wave-tm-extended-detection-and-response-xdr-providers-q4-2021/RES176400.

[214] 6sense. “Identity and access management”. Accessed: July 2024, [Online]. Available: https://6sense.com/tech/identity-and-access-management.

[215] Waterfall Security. “Cybersecurity in the AVEVA enterprise SCADA product – going deep”. Accessed: July 2024, [Online]. Available: https://waterfall-security.com/ot-insights-center/ot-security-standards/cybersecurity-in-the-aveva-enterprise-scada-product-going-deep-jake-hawkes-episode-122/.

[216] M. Russinovich. “Psexec”. Accessed: June 2024, [Online]. Available: https://learn.microsoft.com/en-us/sysinternals/downloads/psexec.

[217] The MITRE Corporation. “T1136.001”. Accessed: June 2024, [Online]. Available: https://attack.mitre.org/techniques/T1136/001/.

[218] T. Pasquier, X. Han, M. Goldstein, T. Moyer, D. Eyers, M. Seltzer, and J. Bacon, “Practical whole-system provenance capture”, in *Symposium on Cloud Computing (SoCC’17)*, ACM, 2017.