International Journal of Information Security (2026) 25:148
https://doi.org/10.1007/s10207-026-01318-x

REGULAR CONTRIBUTION

CAM-LDS: cyber attack manifestations for automatic interpretation of system logs and security alerts

Max Landauer[1] · Wolfgang Hotwagner[1] · Thorina Boenke[1] · Florian Skopik[1] · Markus Wurzen berger[1]

Received: 18 March 2026 / Accepted: 6 August 2026 / Published online: 26 August 2026
© The Author(s) 2026

Abstract

Log data are essential for intrusion detection and forensic investigations. However, manual log analysis is tedious due to high data volumes, heterogeneous event formats, and unstructured messages. Although many automated log-analysis methods exist, they usually still rely on domain-specific configurations, including expert-defined detection rules, handcrafted log parsers, or manual feature-engineering. Crucially, the level of automation of conventional methods remains limited by their inability to semantically understand logs and explain their underlying causes. In contrast, Large Language Models enable domain- and format-agnostic interpretation of system logs and security alerts. Unfortunately, research on this topic is challenging, because publicly available labeled data sets covering a broad range of attack techniques remain scarce. To address this gap, we introduce the Cyber Attack Manifestation Log Data Set (CAM-LDS), a fully open-source and reproducible Linux-based data set for analyzing cyber attack manifestations in system logs and security alerts. Unlike existing datasets that are often network-centric, Windows-focused, or limited in attack diversity, CAM-LDS comprises seven attack scenarios covering 81 techniques across 13 tactics and collected from 18 host- and network-based data sources. We isolate log events temporally associated with individual attack steps to enable analysis of manifestations concerning command observability, event frequencies, performance metrics, and intrusion detection alerts. We further present an illustrative case study utilizing an LLM to process CAM-LDS. The best-performing model ranks at least one correct attack technique first for 41.8% of attack steps and within Top 10 for 67.2%, demonstrating the potential of LLM-based log interpretation and utility of our data set.

Keywords System log data · Cyber attack data set · Log interpretation · Large language models · Intrusion detection · Security alerts

1 Introduction

Protection against the ever-growing increase of cyber attacks is of paramount importance for organizations across all sectors. Among the various domains of cyber security, log data analysis stands out as both highly valuable and inherently complex. On the one hand, system log data presents a persistent and fine-grained record of activities occurring on networks and computer systems. Accordingly, they constitute a fundamental source of information for the detection of imminent attacks as well as forensic investigations of malicious or undesired activities after the fact [1, 47].

On the other hand, log data appear in large volumes, involve heterogeneous formats, and often contain unstructured messages. Their interpretation therefore requires substantial manual effort and expert knowledge [65, 72, 91]. Although automated analysis methods exist, their practical applicability remains limited: signature-based detection approaches rely on manually crafted rules that are difficult to maintain and ineffective against novel and unknown attacks [33]; anomaly-based methods require large amounts of representative training data [44], frequently suffer from high false positive rates [33, 53], depend on manual feature engineering

✉ Max Landauer
max.landauer@ait.ac.at

Wolfgang Hotwagner
wolfgang.hotwagner@ait.ac.at

Thorina Boenke
thorina.boenke@ait.ac.at

Florian Skopik
florian.skopik@ait.ac.at

Markus Wurzen berger
markus.wurzenberger@ait.ac.at

[1] Austrian Institute of Technology, Giefinggasse 4, 1210 Vienna, Austria

[58, 83], often struggle to adapt to evolving attack patterns [58], and typically report statistical deviations without providing semantic explanations of the underlying causes [24, 44, 91].

The emergence of Large Language Models (LLM) has significantly changed how security engineers approach log analysis. Recent studies in Security Operations Centers (SOC) indicate that practitioners increasingly use LLMs to interpret and explain system logs and other low-level telemetry data [23, 75, 76]. Thereby, the key capability of LLMs is their ability to semantically analyze log data in a manner analogous to natural language [32], which supports engineers in understanding unfamiliar log formats or unknown events without manual lookup of documentations [43].

Aside from direct interpretation of log snippets, LLMs have been successfully demonstrated for various log analysis tasks such as log parsing, failure detection, intrusion detection, root cause analysis, and summarization [3, 42, 48]. In contrast to traditional approaches that require separate specialized models for different tasks and log formats, LLMs provide a unified framework capable of supporting or autonomously performing multiple analysis objectives [32, 42]. Consequently, the automation of SOC workflows through LLM-based methods is expected to expand further, even though several challenges remain, including significant computational overhead, high costs, hallucinations, inconsistent outputs, ambiguous terminology, oversimplified reasoning, and limited transparency [23].

A major obstacle for advancing research on LLM-based log interpretation is the lack of suitable public log data sets for evaluation [23]. Researchers therefore often resort to private data sets [9, 72], which hinder reproducibility and comparability, or data sets without security context [32, 42, 43], which limits the relevance of results for security applications. While some log data sets from the intrusion detection domain exist, they typically focus on network traffic [2, 13, 73], are restricted to Windows environments [16–19], or originate from noisy environments in which isolating and labeling attack traces is challenging [36, 61]. Critically, most security-related data sets cover only a limited set of attack tactics and techniques, thereby failing to represent the diversity of real-world kill chains and limiting the scope of evaluations (cf. Table 1 for a comparative overview).

To address these challenges and enable reproducible research in log interpretation and related areas, we introduce the *Cyber Attack Manifestation Log Data Set (CAM-LDS)*, a collection of system log data and security alerts that are direct consequences of cyber attacks. In contrast to prior data set collection strategies, we specifically focus on implementing a broad range of distinct attack techniques and arranging them into multiple coherent kill chains. Thereby, we pursue realistic attack execution within an idle network, allowing us to collect labeled manifestations of attacker behavior while minimizing unrelated background noise. To the best of our knowledge, CAM-LDS is the first publicly available data set specifically designed to support research on LLM-based interpretation of system logs and security alerts.

We release the system log data[1] and network packet captures[2] online. In addition, we provide the virtual testbed infrastructure *AttackBed* together with all implemented attack automation scripts as open-source code.[3] Furthermore, we publish the artifacts required to reproduce our case study for LLM-based log interpretation, including prompts and model responses.[4] We summarize our contributions as follows.

- A labeled data set of cyber attack manifestations comprising system logs and security alerts,
- a systematic analysis and categorization of the collected attack manifestations, and
- a case study demonstrating the utility of the proposed data set for LLM-based log interpretation.

The remainder of this paper is structured as follows. Section 2 summarizes related work on LLM-based interpretation of system logs and security alerts and furthermore reviews existing data sets. Section 3 describes the methodology for generating the data sets introduced in this paper. Section 4 analyzes the captured attack manifestations extracted from the data. Section 5 presents an illustrative evaluation of LLM-based interpretation. Section 6 discusses the implications and limitations of our work and outlines directions for future research. Finally, Sect. 7 concludes the paper.

2 Related work

In this section we first summarize relevant publications in the area of system log and alert interpretation using LLMs. Subsequently, we provide a review of public data sets that are useful to evaluate such approaches.

2.1 LLM-based interpretation

We first discuss approaches that apply LLMs to generic system logs and then focus on work conducted in the cyber security domain.

[1] https://zenodo.org/records/18390560 (accessed: 2026-06-29)

[2] https://zenodo.org/records/18701094 (accessed: 2026-06-29)

[3] https://github.com/ait-testbed/attackbed (accessed: 2026-06-29)

[4] https://github.com/ait-aecid/attack-manifestations-interpretation (accessed: 2026-06-29)

2.1.1 Analysis of generic log data

The rapid emergence and advancement of LLMs have influenced a wide range of research fields, including system log analysis. Since logs are typically available in unstructured or highly heterogeneous formats, LLMs appear as a natural candidate to assist human analysts in extracting actionable insights from such data. One of the earliest systematic investigations in this area is presented by Mudgal et al. [48], who evaluate the extent to which LLMs can infer information directly from raw log data. They consider the following key analysis tasks: *Log Parsing*, which aims to derive log templates and extract parameters, *Log Analytics*, which involves disciplines such as anomaly detection, root cause analysis, and event prediction, and *Log Summarization*, which provides concise summaries of large amounts of log data. In addition to parsing and anomaly detection, the study of Liu et al. [42] considers *Log Interpretation* as the generation of natural-language explanations of the semantic meaning of log messages, as well as *Root Cause Analysis* and *Solution Recommendation* as post-detection activities that aim to support analysts with actionable advice for the resolution of potential issues.

Several recent publications present approaches that automate some or all of these log analysis tasks. Liu et al. [44] propose LogPrompt, a framework for anomaly detection in log data using tailored prompts for ChatGPT. In their subsequent work, the authors propose LogLM [42], a model designed to support all major log analysis tasks within a unified framework. Shanto et al. [72] develop a domain-specific LLM trained explicitly to explain error messages in console logs. Qi et al. [65] introduce LogGPT, which leverages ChatGPT to detect log anomalies in a zero-shot setting, where no training data is provided, as well as a few-shot setting, where labeled sample data is available to the model. While many studies primarily evaluate detection accuracy, Zhang et al. [91] emphasize the practical importance of providing meaningful interpretations alongside detections. To address this gap, they fine-tune a model termed LLM-LADE, which jointly performs anomaly detection and explanation.

Retrieval-augmented generation has recently emerged as another direction for LLM-based log analysis. RAG-based methods retrieve relevant log examples, templates, documentation, or historical incidents and use them to ground the model's output, thereby addressing limitations such as hallucinations, missing domain knowledge, and incomplete context. Starting from early approaches that apply retrieval-augmented language models to log anomaly detection [59], subsequent extensions combine retrieval with log representations to reduce the impact of parsing errors [90], integrate active learning to maintain the quality of the retrieval database [12], and focus on resource-efficient and context-aware usage [89]. Finally, related work on LLM-based online log parsing similarly uses retrieval to provide relevant examples for parsing previously unseen log messages [4, 30].

2.1.2 Analysis of security logs and alerts

All approaches discussed in the previous section are evaluated on data sets that comprise log data corresponding to normal system or user behavior as well as failure or errors logs. A key difference to log data analysis in the security context is that data sets comprise attack traces that result from deliberate and adversarial actions rather than unintentional failures. While some concepts from anomaly detection and interpretation are applicable in both domains, correct classification and reasoning about attack manifestations generally requires to understand an attacker's intent, place each attack step in the context of an attack chain, and incorporate external knowledge from attack frameworks and tools.

Karlsen et al. [32] evaluate LLMs on both generic and security-focused data sets and show that fine-tuning substantially improves detection performance across domains. Additionally, they demonstrate how LLMs can highlight those parts of log entries that contribute most strongly to a classification decision, thereby enhancing model transparency for human analysts. Palma et al. [58] address the high computational and financial costs associated with large proprietary models and instead evaluate a locally deployed language model for log-based detection and interpretation. Sun et al. [78] present a framework for host-based intrusion detection that relies on analysis of raw system logs by LLMs. Their work addresses common issues for such tasks, such as context window limitations that prevent analysis of large chunks of log data.

Beyond log explanation, LLMs have also been applied to map low-level log data to higher-level threat intelligence concepts. Cotti et al. [9] present OntoLogX, which constructs ontology-grounded knowledge graphs from raw logs and uses them to predict MITRE ATT&CK tactics. Tejero et al. [79] demonstrate that LLMs outperform conventional machine learning techniques for the task of attack classification in Internet of Things (IoT) logs. Moreover, they map attack classes to entries of the CAPEC [81] attack pattern database and thereby show that LLMs are able to enrich detections with mitigation recommendations tailored to IoT environments.

The interpretive capabilities of LLMs are also suitable to explain intrusion detection alerts. Balogh et al. [3] employ LLMs to analyze alerts generated by a network intrusion detection system and map them to MITRE ATT&CK techniques. Although some correct classifications are achieved, the authors conclude that the models are currently insufficiently reliable and too resource-intensive for automated deployment in practice. Instead of relying on general-purpose chatbot interactions, Hans et al. [24] achieve higher clas-

sification accuracy with a more specialized pipeline that combines retrieval-augmented mapping to attack techniques and a controlled data set of intrusion alerts from a red-team exercise as well as attacker notes. As part of their work, they also investigate alert grouping, i.e., segmenting continuous alert streams into clusters representing individual attacker actions or attack steps. Zha et al. [88] also address the alert grouping problem by using LLMs to infer the root causes of alerts, specifically to trace cascading effects of service failures that often cause alert floods.

2.2 Data sets

Publicly available log data sets play a central role in the development and evaluation of methods for log analytics, as they enable reproducible experimentation and fair comparison across approaches. Historically, however, high-quality data sets have been sparse, particular those that fulfill common requirements such as correctness, relevance, realism, and timeliness [6, 36]. In recent years, several log data sets have emerged as de facto benchmarks, and a number of newly published data sets constitute suitable candidates for specific research directions, including LLM-based log interpretation. In the following, we review existing system log data sets, compare their key characteristics, and highlight notable properties of individual contributions.

Table 1 summarizes relevant properties of all reviewed data sets, including the types of collected data, the approach to attack generation, the granularity of labels, the operating systems from which logs are captured, the emulated network topology, and the number of covered attack tactics and techniques corresponding to the MITRE ATT&CK framework. For comparability, we only count techniques, because not all data sets provide labels at the sub-technique level; where necessary, sub-techniques are therefore mapped to their corresponding techniques.

Some of the most widely used log data sets, in particular in the research area of anomaly detection, are provided in the LogHub repository [92] as well as the collection of supercomputer logs originally described by Oliner et al. [55]. These log data sets primarily comprise logs from normal system behavior as well as events that indicate failures; however, there are no traces of cyber attacks present in the data. Consequently, their relevance to cyber-security-focused research is limited. Nonetheless, they are suitable for evaluation of generic approaches that leverage LLMs for anomaly detection or log interpretation as presented in Sect. 2.1.1.

Several data sets have been collected specifically for evaluating intrusion detection systems, with a primary focus on network traffic [20]. For example, the authors of PicoDomain [40] simulate an enterprise environment and prepare a kill chain comprising client-side exploitation, privilege escalation, and data exfiltration. Data collection relies on the network monitoring tool Zeek [82], which produces network-centric log files. The CICIDS2017 [73] data set includes both network traffic and system log data; however, since only network traffic is labeled, it is predominantly used for network intrusion detection. A similar limitation applies to DAPT-2020 [49], which collects system logs from various sources such as syslog and audit, but only provides labels for network traffic.

The UWF-ZeekData22 [2] data set was collected during a cyber security exercise in which students employed both defensive and offensive tools. While the labeled data set comprises attacks from all 14 MITRE ATT&CK tactics, the vast majority of events corresponds to reconnaissance activities such as port scanning, and data collection is limited to Zeek logs. UWF-ZeekData24 [13] is a follow-up data set to UWF-ZeekData22 that replaces student-driven attacks with scripted executions, which reduces timestamp inaccuracies, labeling errors, and noise according to the authors.

Beyond network-centric data sets, several data sets that include system logs rely on red teaming exercises for attack execution. The PWNJUTSU data set [5] contains attack traces generated by 22 offensive security experts and includes both network traffic and diverse system logs, such as Linux authentication, access, and audit logs, as well as Windows event logs. The authors of CasinoLimit [34] emphasize that capturing both network traffic and system logs is important to reconstruct attack scenarios. They emulate the network infrastructure of a casino and task 114 penetration testers with attacking the environment. Subsequently, they manually label the resulting data set with 67 distinct attack techniques observed over a time span of 10 h. The authors of COMISET [61] rely on rule-based labeling of attack traces produced by a red team targeting an emulated small enterprise network of Windows hosts, yielding 50 techniques across 11 tactics in two environments.

While red-teaming and penetration-testing campaigns typically provide broad coverage of attack behaviors [13], data sets that focus on specific tactics and settings often rely on scripted attacker behavior. For example, the APT-Persistence data set [66] concentrates on persistence techniques, LMDG [45] focuses on lateral movement, CIDDS-113 [87] centers on file-encryption attacks, and CYSAS-S3 [46] targets a military-oriented use case.

Most existing data sets, however, model attack scenarios that follow a full or partial kill chain and therefore span multiple tactics. AIT-LDSv1.1 [38] includes network scanning, account discovery, brute-force initial access, exploitation for persistence, and privilege escalation, and was designed to support the evaluation of host-based intrusion detection systems. A distinguishing feature of this data set is the collection of fine-grained Linux audit logs. The same authors introduce AIT-LDSv2.0, which includes additional attack vectors, collects logs from multiple hosts, and improves event label-

Table 1 Overview of reviewed data sets

Dataset	Year	Labels	Generation	Number of tactics	Number of techniques	System logs	Network traffic	HIDS alerts	NIDS alerts	Environment	Network
Supercomputer Logs [55]	2006	Failure Type	Real	0	0	✓				Supercomputer	Generic
CICIDS2017 [73]	2017	Attack Type	Scripted	10[a]	19[a]	✓	✓			Windows + Linux	Enterprise
Loghub [92]	2019	Various	Mixed	0	0	✓				Various	Various
Mordor/OTRF [16]	2019	Technique	Scripted	9	35	✓	✓			Windows + Linux	Enterprise
AIT-LDSv1.1 [38]	2020	Attack Type	Scripted	7[a]	15[a]	✓				Linux	Enterprise
CYSAS-S3 [46]	2020	Attack Type	Scripted	8	11	✓	✓	✓	✓	Windows + Linux	Military
DAPT-2020 [49]	2020	Attack Type	Red Team	4	16	✓	✓		✓	Linux	Enterprise
EVTX-ATTACK-SAMPLES [17]	2020	Technique	Scripted	11	58	✓				Windows	Enterprise
PWNJUTSU [5]	2022	None	Red Team	5	13	✓	✓			Windows + Linux	Generic
UWF-ZeekData22 [2]	2022	Technique	Red Team	14	36		✓			Windows + Linux	Generic
AIT-LDSv2.0 + AIT-ADS [36, 37]	2022	Attack Type	Scripted	8	10	✓	✓	✓	✓	Linux	Enterprise
Unraveled [50]	2023	Technique	Manual	5	12	✓	✓		✓	Windows + Linux	Enterprise
PicoDomain [40]	2023	Attack Type	Scripted	8[a]	19[a]		✓			Windows	Enterprise
LMDG [45]	2024	Attack Type	Scripted	5[a]	10[a]	✓	✓			Windows	Enterprise
UWF-ZeekData24 [13]	2024	Technique	Scripted	7	5		✓			Windows + Linux	Enterprise
APT-Persistence [66]	2024	None	Scripted	6[a]	11[a]	✓		✓		Windows	Enterprise
ATLASv2 [71]	2024	Attack Type	Manual	13[a]	30[a]	✓		✓		Windows	Generic
EVTX-to-MITRE-Attack [18]	2024	Technique	Scripted	11	49	✓		✓		Windows	Enterprise
Linux-APT [31]	2024	Technique	Manual	12	26	✓		✓		Linux	Generic
Atomic-EVTX [19]	2025	Technique	Scripted	13	121	✓				Windows	Generic
CasinoLimit [34]	2025	Technique	Red Team	14	67	✓	✓		✓	Linux	Enterprise
COMISET [61]	2025	Technique	Red Team	11	50	✓				Windows	Enterprise
CIDDS-113 [87]	2026	Attack Type	Scripted	5[a]	6[a]	✓	✓	✓		Windows + Linux	Enterprise
CAM-LDS (Our Dataset)	2026	Technique	Scripted	13	81	✓	✓	✓	✓	Linux	Enterprise

[a] Tactic and technique counts are not explicitly stated in the corresponding publication and are thus estimated based on attack descriptions

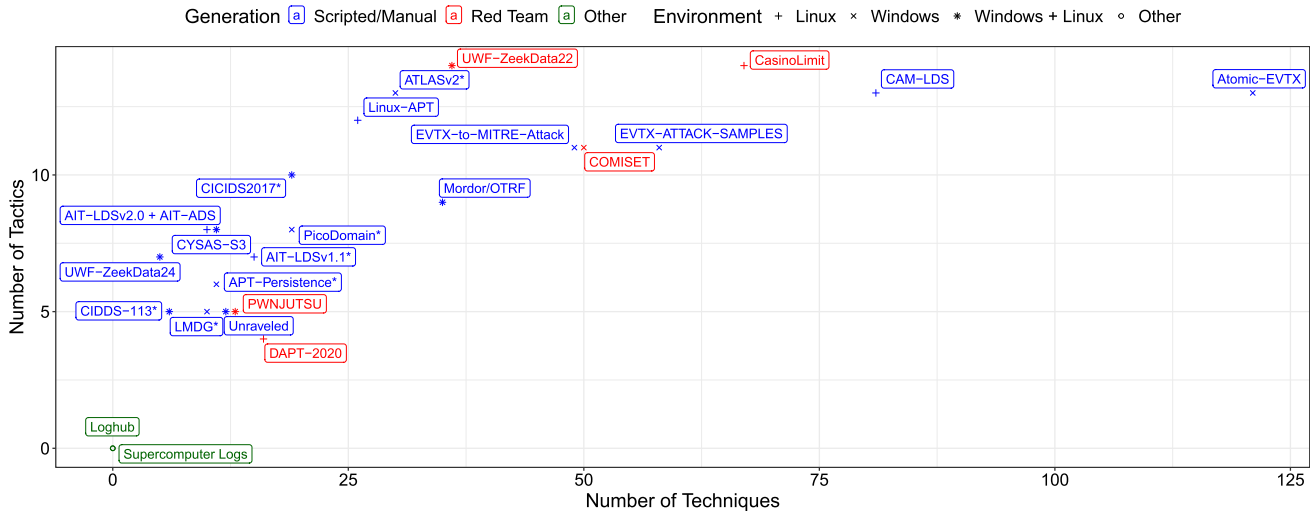


Fig. 1 Number of MITRE ATT&CK tactics and techniques covered by the reviewed data sets. Asterisks indicate that tactics and technique counts are not explicitly stated in the paper and thus estimated based on attack descriptions

ing. In another follow-up publication, the authors present AIT-ADS [37], an alert data set generated by forensically processing the AIT-LDSv2.0 with signature- and anomaly-based intrusion detection systems. The attack chain emulated in the ATLASv2 [71] data set relies on phishing for initial access and aims at the exfiltration of documents. Captured data sources involve audit logs, browser application logs, DNS logs, and security alerts from a host-based intrusion detection system. Notably, the authors emphasize manual execution of the attack steps to enhance realism.

While the aforementioned data sets employ descriptive attack type labels, several other data sets adopt tactic- and technique-level labeling aligned with the the MITRE ATT&CK framework. This includes several publicly available data sets not directly associated with research publications, such as Atomic-EVTX [19], EVTX-to-MITRE-Attack [18], and EVTX-ATTACK-SAMPLES [17]. While these data sets cover a large number of techniques, they are restricted to Windows environments. The Mordor/OTRF [16] data set extends coverage to Linux hosts and web services in addition to Windows event logs. Unraveled [50] builds upon DAPT-2020 [49] by expanding attack tactics, improving IDS deployment, and refining labels. Linux-APT [31] simulates the behavior of selected APT groups and provides technique-level labels of attacks against Linux-based environments.

In summary, several existing data sets are ill-suited for log interpretation in a cyber-security context, either because they do not contain attack traces or because they focus primarily on network traffic rather than system logs. As illustrated in Fig. 1, many data sets cover fewer than nine tactics and fewer than twenty techniques, which constrains the scope of evaluations conducted on them. Data sets with broader coverage of attack techniques typically originate from red teaming exercises, which introduces challenges related to controllability, reproducibility, and labeling reliability [13], or are limited to Windows-only environments, thereby excluding Linux-specific log sources.

To address these limitations, we introduce CAM-LDS, a system log data set collected through seven fully scripted attack scenarios that cover 81 distinct attack techniques executed in a reproducible Linux environment. Accordingly, we consider it as a complementary data set to existing Windows-centric data collections. In addition to system logs and network traffic, CAM-LDS includes alerts generated by host- and network-based intrusion detection systems, as well as system configuration data that provides valuable contextual information for log interpretation and attack analysis.

3 Data set generation

This section describes our approach to generate the CAM-LDS. After explaining our methodology, we present the network topology common to all scenarios, the adversarial emulation tool used to execute the attacks, and the corresponding attack chains of each scenario.

3.1 Methodology

In this section, we describe our methodology for generating log data sets. We thereby refer to the Steps (1)–(9) displayed in Fig. 2, which summarizes our methodology.

3.1.1 Scenario design

Our methodology is driven by the objective of collecting log data corresponding to a wide variety of attack techniques. Accordingly, Step (1) of our methodology focuses

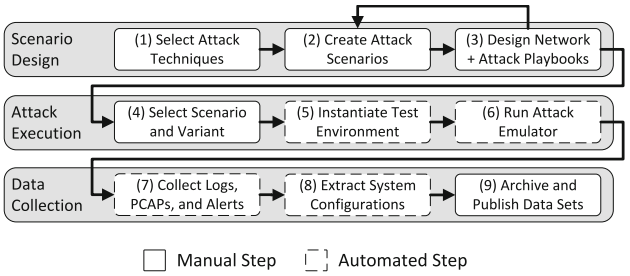


Fig. 2 Overview of our methodology

on the systematic selection of suitable techniques from the MITRE ATT&CK framework, one of the most widely adopted and up-to-date references for modeling adversary behavior. At the time of writing, the ATT&CK Enterprise matrix comprises 216 distinct techniques.[5] We review each technique individually and assess it according to the following criteria: (i) *Observability*. This criterion is intended to exclude techniques whose execution would leave the collected logs effectively unchanged. To make the selection process systematic, we base our selection on the "Detection Strategies" specified for each technique in the MITRE ATT&CK framework. These recommendations describe how adversary behavior can be detected in practice and often indicate relevant data sources. We therefore focus on techniques whose detection strategies suggest the generation or modification of observable artifacts in system or application logs. Techniques that do not produce such traces are of limited relevance for log-centric analysis and are consequently excluded. For example, we exclude most techniques associated with the Resource Development tactic, which primarily describes activities that occur outside of the target infrastructure, such as acquiring domains or social media accounts. (ii) *Feasibility*. We restrict our selection to techniques that can be realistically emulated within our Linux-based test environment. As a result, techniques that are specific to other platforms, such as the Windows operating system, various mobile systems, or cloud-native environments, are excluded. (iii) *Automation*. We require that execution of the attack technique can be fully automated to support scripted attack chains and ensure reproducibility. Consequently, techniques that inherently involve manual interaction with physical hardware, such as inserting USB devices, are omitted.

After applying these criteria to the entire framework, we end up with a set of 122 attack techniques for potential implementation. Thereby, in order to construct a challenging data set where some individual attack steps are only recognizable as such when considered in the context of the entire kill chain, we purposefully keep techniques that could yield manifestations similar to those of benign system behavior. In particular, this includes activities commonly performed by administrators, such as system information collection.

Step (2) aims to create attack scenarios based on the initial set of 122 attack techniques. We start by grouping techniques according to their infrastructure requirements, such as the need for specific services or particular network configurations. Subsequently, we arrange the grouped techniques into coherent and realistic attack sequences, following common kill chain models and reflecting the complexity of real-world attacks in terms of technique diversity [74], ultimately resulting in seven attack scenarios. At this stage, we exclude some techniques that cannot be meaningfully integrated into any scenario. We further observe that several techniques are functionally interchangeable, and therefore introduce variants in our scenarios, where individual attack steps are realized using different techniques while achieving the same attacker objective. After completing this step, we obtain a conceptual specification of both the required network infrastructure and the complete set of attack scenarios.

The technical realization of the infrastructure and the implementation of the attack emulation playbooks in Step (3) is carried out as an iterative task that is coupled with Step (2). The reason for this is that during development and testing, techniques may be moved between or duplicated to other scenarios to produce more informative or diverse log manifestations. Moreover, certain techniques that turn out impractical or unsuitable for the environment or attack chains are removed, and the scenarios are revised accordingly. Note that during this phase we also carry out validation activities to ensure that all scenarios are logically consistent, correctly implemented, and accurately labeled with the corresponding attack techniques. Thereby, all attack playbooks and scripts are independently cross-checked by two other experts. After completion of the implementation phase, we eventually end up with a fully scripted infrastructure as well as playbooks for 81 attack techniques across seven scenarios.

3.1.2 Attack execution

In Step (4), we select one of the previously designed attack scenarios, which are accompanied by configuration files that specify the required network topology as well as the necessary software components and versions. Note that the subsequent steps refer to a single simulation run; however, we repeat these steps for every scenario and for all possible combinations of scenario variants. Step (5) instantiates the infrastructure for the selected scenario in a virtualized environment, which is done from scratch for every scenario to avoid that any traces left from previous attacks remain on the systems and to ensure that the infrastructure is in a predictable state. Deployment of the network and all hosts is fully automated, as the entire infrastructure is specified using Infrastructure-as-Code. We employ OpenTofu [57] and Ter-

[5] This work relies on version 18.1 of the MITRE ATT&CK Enterprise matrix, which is the most recent version available at the time of writing.

ragrunt [22] to provision the network components and virtual machines, and use the configuration management framework Ansible [70] to install and configure all software components on the deployed hosts.

Step (6) executes the attack emulation for the selected scenario and, where applicable, its variants. Note that we do not start the emulation tool immediately after infrastructure deployment, as newly instantiated systems may generate background log events during and shortly after startup. To prevent such events from overlapping with attack traces, we introduce a stabilization period of 15 min before starting the attack emulation; this value was determined empirically during test runs and found sufficient for our test environment. In addition, log events may be generated with slight delays following individual attack actions, for instance, due to communication overheads or buffering when large numbers of events are generated. To account for this, we enforce a minimum sleep interval of 15 s between consecutive attack steps.

3.1.3 Data collection

After completion of the attack execution, Step (7) collects log data from 18 distinct data sources and across all hosts in the environment. Table 2 provides a list of all available data sources. This includes standard Linux log sources such as audit logs, system logs (syslog), authentication logs, package management logs, and CRON logs. In addition, we collect application-specific logs from services deployed in the scenarios, including Apache access and error logs, Zoneminder, FTP, Puppet, Docker, Nextcloud, and the mail server. System performance metrics are captured through collectd [8], which continuously records system utilization at short intervals. We further collect alerts generated by the host-based intrusion detection system Wazuh [86], which we utilize with its default rule set. Netflows and network packet captures are captured using Suricata [56], which simultaneously serves as our network-based intrusion detection system and is also used with standard detection rules. We use the default rule sets of Wazuh and Suricata to reflect a commonly available out-of-the-box deployment and avoid dataset-specific tuning that could bias the generated alerts toward the implemented attack scenarios.

Step (8) extracts system configurations from all hosts. These configurations provide important contextual information for forensic analysis, such as determining the presence of vulnerable software versions or insecure system settings. In the final Step (9), we archive all collected data and release them in their raw format without modification to ensure that no potentially relevant information is lost. Anonymization is not required, as no human users interact with the infrastructure during the simulations.

3.2 Network topology

Cyber attacks rarely affect a single host in isolation; instead, their traces typically manifest across multiple interconnected systems. Gaining a comprehensive understanding of attacker behavior, e.g., as part of forensic investigations conducted after an incident, therefore requires correlating log data from multiple hosts and infrastructure components. To capture attack traces from a realistic setup of such a distributed systems, we design a network topology that approximates the IT infrastructure of a small enterprise as illustrated in Fig. 3.

Following established best practices for network segmentation [28], the topology is divided into five security zones that are interconnected via a firewall: (i) *Internet*. This zone provides external connectivity and contains a public DNS server as well as a corporate DNS server for the simulated organization. In all but one of our scenarios, the attacker gains initial access through the Internet zone. (ii) *Demilitarized Zone (DMZ)*. The DMZ hosts several publicly exposed services, including a Docker server running a mail platform and a cloud service as containers, a repository server used for internal asset management, and a video server providing video surveillance functionality. (iii) *Local Area Network (LAN)*. The LAN provides internal services to the organization, including a file share accessible to employees. (iv) *User Network*. This zone represents the standard working environment of employees, who access the enterprise network using their company workstations. (v) *Admin Network*. The admin network is reserved for network administrators and serves as the default zone for privileged access to the infrastructure.

The network is instantiated on a per-scenario basis, deploying only those components that are required for executing the attack and for collecting the corresponding traces. Figure 3 shows which components are deployed in each scenario by referencing the number of the respective scenario in blue circles. For example, in Scenario 5 we do not deploy the file share, but assume that a network administrator is connected to the LAN. In this scenario, the attacker also accesses the network through the LAN rather than the Internet. The next section describes our method for emulating the attacker's behavior in detail.

3.3 Attack emulation

Our attack emulation strategy is guided by two primary requirements: repeatability and realism of the generated log artifacts. Repeatability is essential to enable multiple executions of identical attack scenarios, which in turn allows to verify the consistent generation of attack manifestations across runs. Moreover, repeatability enables automation of attack execution, thereby reducing manual effort during data collection and minimizing inconsistencies as well as human

Table 2 Overview of collected data sources

Data source	Type	Description
Audit logs	System log	System calls and process activity
Syslog	System log	General system messages
Authentication logs	System log	Login attempts and authentication events
Kernel logs	System log	Kernel-level messages
CRON logs	System log	Scheduled task execution
Package management logs	System log	Package installation and update events
Apache access logs	Application log	Web requests to Apache services
Apache error logs	Application log	Web-server errors and warnings
ZoneMinder logs	Application log	Video-server application events
FTP logs	Application log	File-transfer activity
Puppet logs	Application log	Configuration-management activity
Docker logs	Application log	Container-related events
Nextcloud logs	Application log	Cloud-service application events
Mail logs	Application log	Mail-server activity
Collectd logs	Performance metric	CPU, memory, disk, and network utilization
Wazuh alerts	Intrusion alerts	Host-based intrusion alerts
Suricata alerts	Intrusion alerts	Network-based intrusion alerts
Suricata flows/PCAPs	Network traffic	Network flows and packets

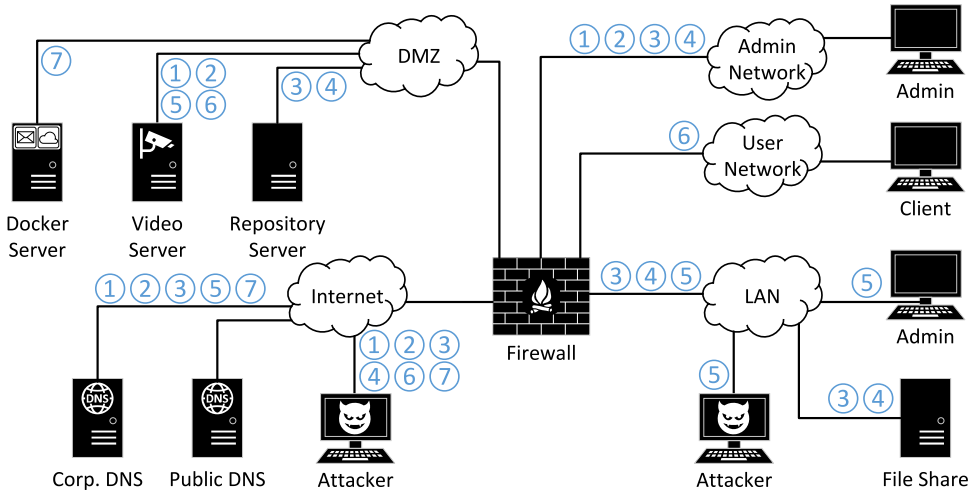


Fig. 3 General network topology and scenario-specific deployments of components

error. Furthermore, automated and repeatable attack scenarios facilitate the systematic generation of variants in which individual steps of an attack chain are replaced or modified, allowing us to cover additional attack techniques in a controlled manner. Since neither red teaming exercises nor manual attack execution (cf. Sect. 2.2) enable repeatable and automated experiments that yield consistent and reproducible results [13], we opt for script-based attack execution.

Naive automation through scripts can negatively affect the realism of observed attack traces [35, 71]. Specifically, it is non-trivial to realize system interactions with simple shell scripts in such a way that they are indistinguishable from actual attacker behavior. For example, human users generally rely on interactive programs such as text editors to modify files, while scripts are limited to non-interactive mechanisms such as file replacement and stream editing to achieve equivalent actions. These differences are visible in system logs and may lead to artifacts that are uncharacteristic of real-world attacks, thereby reducing the authenticity of the resulting data sets and possibly affecting the validity of any evaluation results.

To reconcile the requirements of repeatability and realism, we employ AttackMate,[6] an open-source adversarial emulation framework designed specifically for reproducible experiments with realistic attack execution across all phases of the kill chain. The framework is described in detail in a

[6] https://github.com/ait-testbed/attackmate (accessed: 2026-06-29)

```
vars:
 $MAIL_HOST: mail.attackbed.com
 $DOMAIN: attackbed.com
 $DNS_LIST: /usr/local/share/SecLists/Discovery/DNS/subdomains-top1million-5000.txt

commands:
 - type: shell
   cmd: dnsenum -f $DNS_LIST --dnsserver 192.42.0.233 $DOMAIN
   metadata:
    techniques: "T1590.002, T1591"
    description: "Enumerate subdomains of corporate domain-zone"

 - type: shell
   cmd: ./smtp-user-enum.pl -M VRFY -U smtp-user.txt -t $MAIL_HOST
   metadata:
   techniques: "T1589.002"
    description: "Enumerate email addresses on SMTP-Host"

 - type: shell
   cmd: hydra -l "alice@$DOMAIN" -P smtp-pass.txt -c 5 -s 143 $MAIL_HOST imap
   metadata:
    techniques: "T1078.002, T1110.001, T1133"
    description: "Brute-force IMAP-password using the already enumerated username"
```

Fig. 4 Sample AttackMate playbook for DNS enumeration, user identification, and brute-force login

companion paper [35]. AttackMate uses fully scripted playbooks to define attack chains in a deterministic and repeatable manner. Figure 4 shows an example playbook consisting of three sequential attack steps. Each step executes a predefined command on the target system, yielding consistent log traces as long as the underlying infrastructure remains unchanged. During execution, AttackMate generates a timestamped output file that associates each command with the corresponding attack labels specified in the metadata fields. These time-based labels enable precise correlation between attack steps and collected log data or network traffic.

At the same time, AttackMate preserves realism by natively supporting interactive sessions and sending individual keystrokes that are indistinguishable from manual typing. Unlike many existing adversarial emulation tools that rely on agents installed on target hosts, AttackMate operates exclusively from a dedicated attacker machine and interacts with the environment solely through remote interfaces such as SSH, thereby accurately modeling an external adversary. In addition to shell command executions, AttackMate further provides dedicated modules for common attack vectors and interfaces with offensive tools widely used by adversaries, such as Metasploit [69].

3.4 Scenarios

Based on the infrastructure setup described in Sect. 3.2, we implement seven attack scenarios as AttackMate playbooks. The scenarios and their respective steps have been designed to represent realistic kill chains that cover many distinct techniques from the MITRE ATT&CK framework. Table 3 provides a concise overview of our seven scenarios, including their title, a description summarizing the overall setup and the main attack vectors, the number of variants (#Variants), the resulting number of simulated attack paths (#Sim.), and the number of labeled attack steps (#Steps). The table specifies variants together with the affected attack phase, e.g., *6 (PVE)* denotes six privilege-escalation variants. In total, our data set comprises 34 simulations and 243 attack steps across all scenarios.

Subsequent sections describe each scenario in detail and include visualizations of the corresponding attack paths. In these state chart diagrams, each block represents one or more attack steps associated with a specific set of technique labels; individual steps correspond to single attacker actions and are identified by the numbers shown within the block. Note that even though execution of some attack techniques only requires a single step, several techniques are realized through sequences of steps and thus grouped for brevity. Furthermore, we indicate the tactics corresponding to the involved techniques using markers above each block, namely Reconnaissance (RCN), Resource Development (RSD), Initial Access (IA), Execution (EXE), Persistence (PST), Privilege Escalation (PVE), Defense Evasion (DEF), Credential Access (CRD), Discovery (DSC), Lateral Movement (LAT), Collection (COL), Command and Control (CNC), Exfiltration (EXF), and Impact (IMP).

3.4.1 Scenario 1: video server exploit

As depicted in Fig. 5, Scenario 1 follows the compromise of a vulnerable video server. After gaining initial access, the attacker escalates privileges, establishes persistence, and performs post-compromise discovery.

The scenario starts with reconnaissance, which is the earliest stage of the kill chain. We do not assume that the attacker has any prior knowledge about the target network beyond its public domain *attackbed.com*. The attacker's first objective is information gathering, which we subdivide into several steps. In the first step, the attacker enumerates subdomains of the corporate DNS server using *dnsenum* [85], which reveals *video.attackbed.com*, *fw.attackbed.com*, as well as *repo.attackbed.com*. The attacker then carries out network mapping of the video subdomain with *nmap* [21], which is useful to discover available services on hosts, and identifies that port 80 (HTTP) is open on the video server. Next, a network vulnerability scan with *nikto* [7] shows that the server

Table 3 Overview of attack scenarios

ID	Name	Description	#Variants	#Sim.	#Steps
1	Video Server Exploit	In addition to some reconnaissance and scanning attacks, this scenario revolves around compromising the video server. A key element in this attack chain is the exploitation of the vulnerable video server software. Subsequently, the attack graph involves six variants that allow the attacker to escalate to root privileges as well as three variants that provide the attacker with persistent access; we simulate all combinations of these variants	6 (PVE) 3 (PST)	18	56
2	Linux Malware	The malware considered in this scenario are an implant that establishes reverse-shell connection as well as a rootkit that targets dynamic linking. The remainder of the scenario broadly covers discovery techniques and exfiltrates some data from the compromised machine	2 (PST)	2	29
3	Lateral Movement	After logging into the firewall through brute-force password guessing, the attacker moves laterally to the file share in the LAN segment. This is achieved by injecting malicious code either into a shared service file, a software management tool, or a manipulated package. The scenario ends with several destructive activities on the target host, including data encryption and deletion. The entire scenario is implemented for both SSH and VNC connections	2 (IA) 3 (LAT)	6	60
4	Network Attack	This scenario aims to compromise the firewall of the network. A port knocking sequence is used to download and run implants, which in turn enable the attacker to modify firewall rules and reach even more hosts in the network	–	1	22
5	Network Sniffing	In this short scenario, the attacker captures authentication tokens from network traffic and re-uses them to gain access to a system	–	1	3
6	Attack on Client	This scenario considers attack cases where a human user is deceived into compromising their own computer, either by downloading and opening a malicious document, installing a fake browser plugin, or providing the attacker access via screen sharing and remote control software. Once inside the system, the attacker installs a backdoor for permanent access. The scenario also involves several activities for data collection, including a keylogger, a data exfiltration tool, and extraction of clipboard data	3 (IA) 2 (PST)	5	60
7	Docker Attack	In this scenario, the attacker first carries out some scanning activities and then brute-forces a password of a mail service. By re-using the collected credentials on a cloud storage service, the attacker is able to execute an authenticated exploit that grants them access to the containerized environment that hosts both the mail and cloud storage services. Once inside, the attacker exploits weak configurations to escape the container and obtain root access on the host	–	1	13

runs Apache and that the main application entry point is *video.attackbed.com/zm/*, which suggests the presence of the open-source video surveillance software ZoneMinder [93]. The attacker then performs deeper web content discovery through fuzzing with *ffuf* [14], a fast web fuzzer that enumerates paths using a wordlist.

Since fuzzing does not yield any direct ways to compromise the system, the attacker searches public databases for recent ZoneMinder vulnerabilities and identifies CVE-2023-26035 [52], which describes a vulnerability in ZoneMinder's snapshot action that enables unauthenticated remote code execution. The attacker then resorts to the Metasploit framework, where they find and execute an already implemented exploit for that vulnerability.

With the ability to execute arbitrary commands on the video server, the attacker configures an interactive shell and pursues their new objective of escalating their privileges. To enable fast enumeration of potential privilege escalation vectors, the attacker uploads and executes *linpeas* [60], a vulnerability scanner for the Linux operating system that

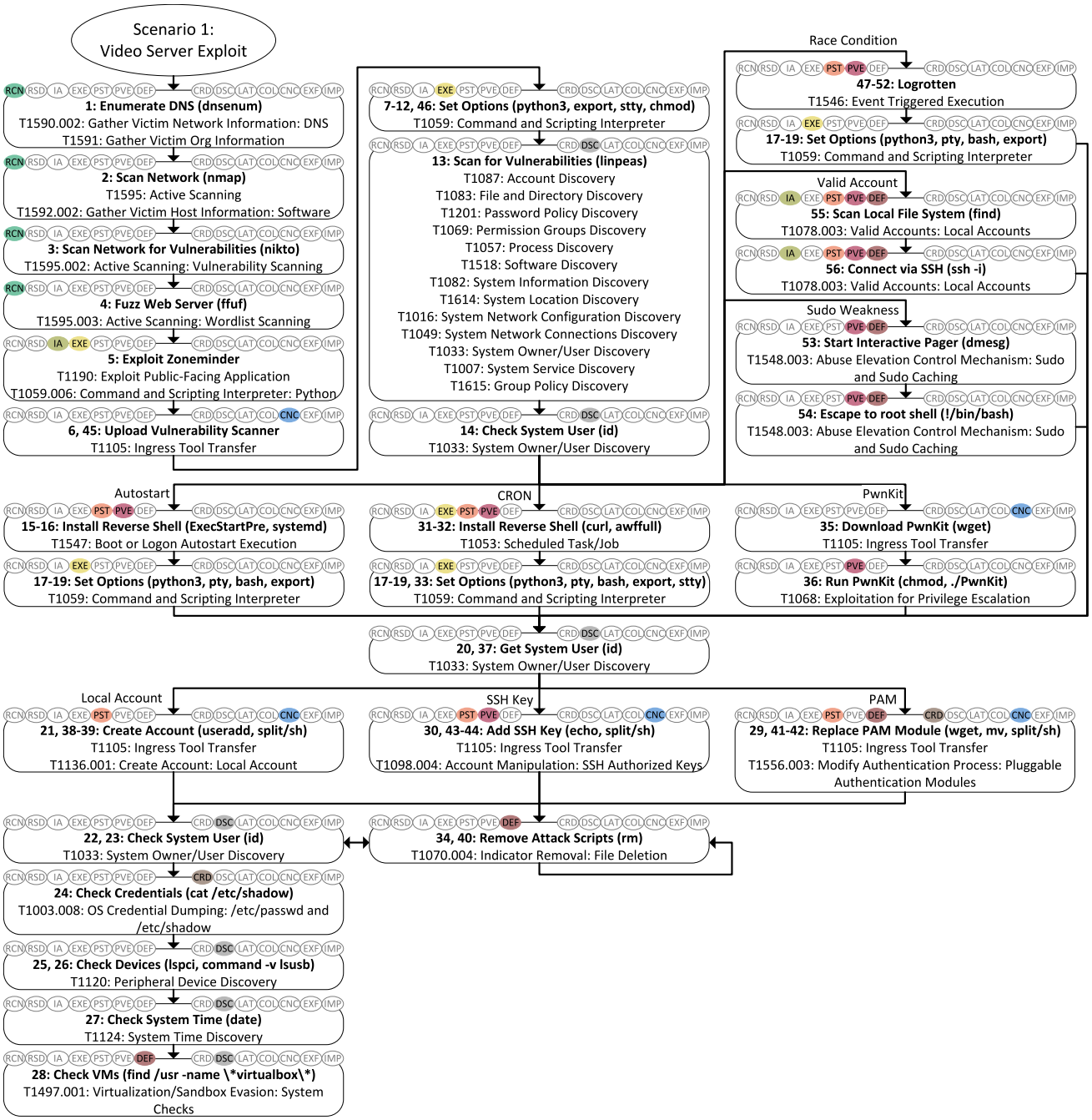


Fig. 5 Attack paths of scenario 1: video server exploit

reports misconfigurations, weak permissions, and other common escalation paths as indicated by the broad coverage of attack techniques in Fig. 5. After executing the *id* command to record current user context, the attacker proceeds to escalate their privileges through one of six possible variants modeled in Scenario 1, some of them based on the linpeas output. (i) *Autostart*. The attacker uploads a reverse-shell, opens the ZoneMinder systemd unit file with an interactive text editor, and adds a line that executes the payload on service start, e.g., when the system is booted. When the unit is started, the payload is executed and the attacker gains root access through the reverse-shell. (ii) *CRON*. In this variant the attacker injects commands that download and run a reverse-shell into a system utility that is executed periodically by CRON. For the purpose of Scenario 1, we assume that the *awffull* [11] tool, a log analysis script that runs under a privileged account and is managed by CRON, was misconfigured by system operators during installation.

In particular, vulnerable permissions of the corresponding CRON-job allow unprivileged users to inject commands that are subsequently executed with root permissions. Accordingly, as soon as CRON invokes the manipulated script, the injected commands establish the reverse-shell and provide the attacker with root access to the system. (iii) *PwnKit*. The attacker downloads and runs the *PwnKit* [64] script, which is a self-contained exploit for *polkit's pkexec utility* [51] that provides root access. (iv) *Race Condition*. Most Linux distributions come by default with *logrotate* [41], a service that enables automatic rotation, compression, and retention of log files. Unfortunately, this tool is vulnerable to a race condition when it runs under root privileges but the managed log files are located in directories accessible to unprivileged users. In this variant, the attacker makes use of the *logrotten* [27] exploit, which replaces one of the log files with a symbolic link to an attacker-controlled directory just at the right time so that logrotate creates the new log file in the linked directory. Since the attacker has write access to the file managed by a root process, they can use it to inject a reverse-shell. (v) *Sudo Weakness*. This variant assumes that due to a system misconfiguration, non-privileged users are able to execute *dmesg*, which is a utility to print the kernel ring buffer. By running *dmesg* interactively and abusing its shell-escape features, the attacker is able to spawn a root shell. (vi) *Valid Account*. In this simple variant, the attacker scans the local file system and thereby recovers a private SSH key belonging to a high-privileged user account, which they use to establish a privileged SSH session.

After completing each variant for privilege escalation, the attacker runs the *id* command to confirm their newly obtained privileges. At this stage of the attack chain, the attacker's objective is to obtain persistence to retain access to the system even if the previously exploited vulnerabilities are patched, e.g., when system operators upgrade ZoneMinder to a more recent version that is not vulnerable to the exploit. Scenario 1 involves three variants for persistence that we describe in the following. (i) *Local Account*. The attacker creates a new local user with the name *webadmin* to not trigger any suspicions. Then, the attacker transfers their own SSH key to the authorized keys. (ii) *SSH Key*. This variant is similar to the previous one, with the exception that the attacker copies their own SSH key into the authorized keys of an already existing privileged user. (iii) *PAM*. The attacker replaces the system's *Pluggable Authentication Module* (PAM) with a modified variant that provides backdoor access.

Note that for some of these variants, the attacker makes use of the split command to obfuscate their activities and reduce traceability. However, since execution of the split command depends on the type of root shell, we are only able to model this optional technique as part of the *PwnKit*, *Sudo Weakness*, and *Valid Account* variants for privilege escalation. All other variants execute the aforementioned commands directly.

After establishing persistence, the attacker runs the id command once more and removes some of the files downloaded as part of the previous attack steps. The attacker then performs targeted post-compromise discovery, in particular, extraction of credentials and hashed passwords for offline cracking, enumeration of connected peripheral devices, checking of system time and timezone, and searching for virtual machines or hypervisor artifacts.

3.4.2 Scenario 2: linux malware

Figure 6 visualizes an attack chain in which an already compromised Linux server is further infiltrated through malware deployment, followed by discovery, credential collection, and data exfiltration. We assume that the attacker has already obtained root privileges before the start of the scenario, e.g., through an attack chain comparable to Scenario 1. The attacker's immediate objective in the beginning of Scenario 2 is thus to compromise the system further through the installation of malware, which we model in two variants. (i) *CRON*. The attacker connects to the target machine via SSH, starts a HTTPS listener, and transfers an implant via SFTP. The implant is installed by modifying the *awffull* script that has already been targeted in the *CRON* variant of Scenario 1; however, instead of editing the file with a text editor, the attacker uses a stream editor to inject a command that starts the implant. (ii) *Rootkit*. The attacker again connects via SSH, prepares a rootkit, and transfers it via SFTP. Analogous to the previous variant, the attacker also transfers an implant and starts a HTTPS listener. The implant is installed by editing ZoneMinder's systemd unit file similar to the *Autostart* variant of Scenario 1. The rootkit is installed through a dynamic-linker preload technique so that it is loaded system-wide into dynamically linked processes [77]. Finally, the attacker reloads systemd and restarts ZoneMinder's service to trigger execution of the implant through systemd.

After completion of either variant, the attacker's subsequent objective is to collect and exfiltrate relevant information. The post-exploitation phase begins with discovery operations: enumeration of files, running processes, and active network connections, as well as collection of network interface configuration. The attacker then acquires artifacts for offline analysis, including process memory snapshots and a process identifier for the database service. Subsequently, the attacker extracts stored authentication material such as hashed credentials. Collected data is aggregated into archives and extracted; the attacker also removes local artifacts that would reveal the exfiltration event. In addition to bulk collection, the attacker selectively downloads individual files and directories of interest. To support further lateral discovery, the attacker deploys *nmap* that was already used in Scenario

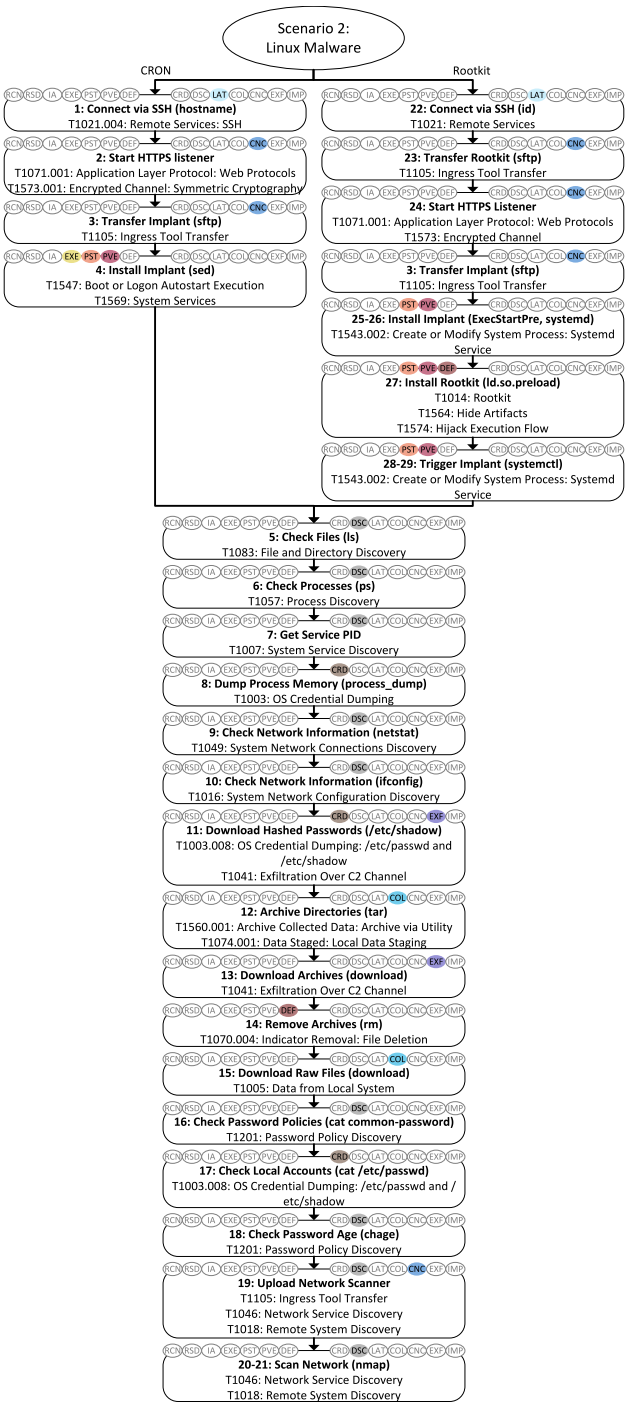


Fig. 6 Attack paths of scenario 2: linux malware

1 and runs it from the compromised host to enumerate reachable hosts and services.

3.4.3 Scenario 3: lateral movement

Scenario 3 emulates how an attacker moves within the network from one host to another, in particular from the initially compromised puppet server to the file share within the LAN segment. After reaching the target host through shared infrastructure, the attacker performs a sequence of destructive actions.

Figure 7 visually summarizes the entire scenario, which involves two variants regarding the way how the attacker connects to the targeted network. (i) *SSH*. The attacker attempts to log into the puppet server via SSH by brute-forcing several combinations of user names and common passwords. To this end, the attacker employs *hydra* [25], a tool that automatically and rapidly goes through a pre-defined file of login credentials and tests whether some of them provide access. For the purpose of our scenario, we assume that one of the users with root privileges has a weak password contained in the list. After logging into the Puppet server, the attacker deploys the network sniffer *tcpdump* to capture credentials transmitted over the network. In our scenario, a scheduled CRON job on the file share periodically retrieves data from the Puppet server via FTP. The attacker intercepts the credentials contained in one of these connections and subsequently re-uses them to escalate to root privileges. (ii) *VNC*. In this variant, the attacker attempts to brute-force a connection via Virtual Network Computing (VNC) rather than SSH. Again, we assume that the attacker manages to guess a weak password of a privileged user after a few attempts, allowing them to log into the puppet server.

Both of the aforementioned variants continue with the attacker printing hashed passwords for offline cracking. Subsequently, they check for the presence of private keys and find one corresponding to the puppet server. Then, the attacker locates a CRON job called *healthcheck* in a Network File System (NFS) share. At this point, we consider several variants how the attacker exploits shared system content to move laterally to another server in the network. Note that while conceptually identical, the technical implementations of these steps are realized differently depending on whether the attacker accesses the system through SSH or VNC as determined by the first variants of the scenario. (i) *Healthcheck*. The attacker edits the *healthcheck* CRON job and injects a line that downloads and establishes a reverse-shell. Once this script is automatically executed on the file share as part of normal *healthcheck* activity, the attacker gains access on that system and has thus achieved lateral movement. (ii) *Puppet*. The attacker realizes that Puppet [62], a service that automates administrative tasks such as server configuration in networks, is installed on the firewall and thus likely used across multiple nodes in the network. The attacker then edits parts of puppet's code; in particular, they inject a malicious class that downloads and executes a reverse-shell and assign that class to the node corresponding to the file share. Next time the Puppet service running on the file share executes that code, the reverse-shell triggers and provides the attacker with access to that system.

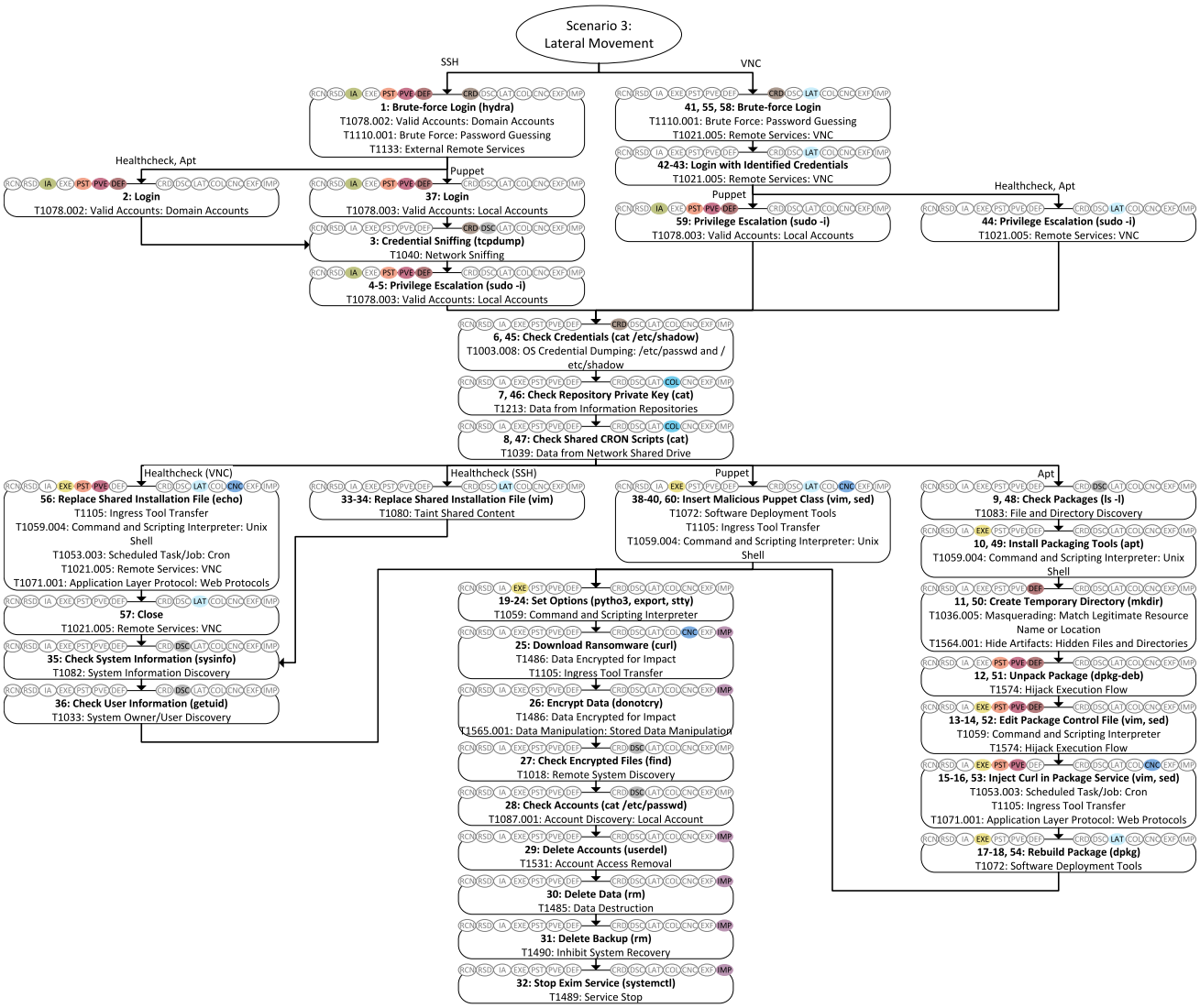


Fig. 7 Attack paths of Scenario 3 lateral movement

(iii) *Apt*. The attacker realizes that the puppet server provides APT packages to other servers, including the file share that updates the *healthcheck* package through puppet. They then decide to build a malicious version of that package that appears as an update. This is achieved by downloading packaging tools, unpacking the *healthcheck* package, adding a malicious CRON job that establishes a reverse-shell to the service, and rebuilding the package. As soon as the admin user triggers the installation of the package, e.g., as part of normal updating procedures, the injected code establishes a reverse-shell and thus allows the attacker to move laterally to the file share. Note that we also emulate administrator activities analogous to attacker behavior to ensure that these attack steps succeed.

After completing any of the aforementioned variants, the attacker first configures the shell for interactive access and then continues with a sequence of destructive attack techniques. First, the attacker downloads a ransomware from their own system and uses it to encrypt the media directory on the system. After checking that the files are indeed encrypted, the attacker enumerates all user accounts available on the system and deletes the account of the user whose password is brute-forced at the beginning of this scenario. The attacker proceeds with deleting some directories as well as the system's backup files. Finally, the attacker stops a system service related to mail exchange.

3.4.4 Scenario 4: network attack

In Scenario 4, we assume that the attacker has already compromised the repository server within the DMZ and seeks to pivot into the internal network by targeting the firewall and modifying its configuration. The attack chain depicted in Fig. 8 starts with the preparation of an implant and an HTTPS

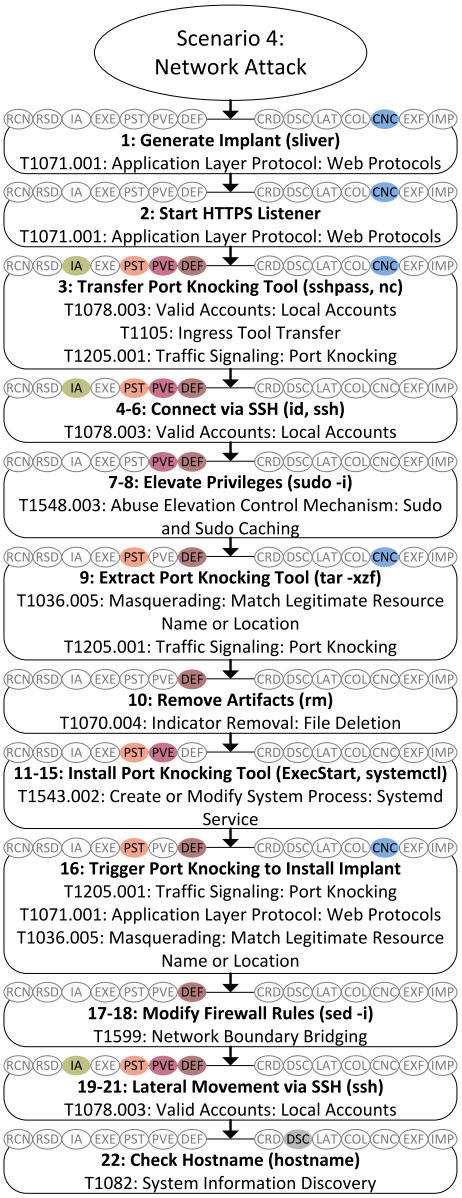


Fig. 8 Attack paths of scenario 4: network attack

listener, similar to the initial steps of Scenario 2. The attacker then re-uses the credentials of a compromised account on the repository server to first transfer a port knocking script to the firewall and then log in as a privileged user. The port knocking script is extracted and installed under the inconspicuous name *auditf* into the location where system binaries are stored; the archive is removed after extraction to reduce forensic traces. The attacker installs the port knocking tool by creating a service that triggers the script on start. Subsequently, they start the service before exiting the firewall and returning to the repository server.

From there, the attacker initiates the port knocking sequence, which causes the previously deployed port knocking tool on the firewall to download and run the implant

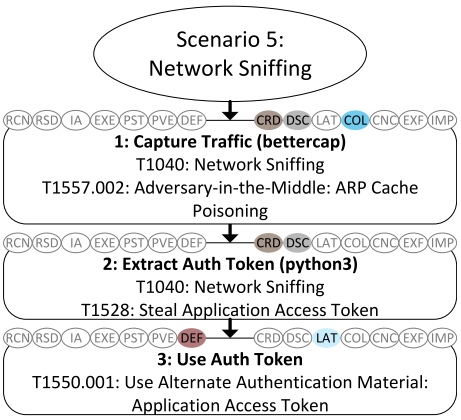


Fig. 9 Attack path of scenario 5: network sniffing

prepared in one of the earlier steps. Through that implant, which provides remote command execution, the attacker then modifies and reloads the firewall rules so that hosts located in the DMZ are able to connect to hosts in the LAN, thereby effectively enabling to access the fileshare from the repository server. The attacker immediately leverages this ability and connects to the fileshare via SSH, where they print some system information to confirm that they reached the intended target.

3.4.5 Scenario 5: network sniffing

Figure 9 visualizes the interception and reuse of an administrator's authentication token through a covert network access device deployed in the LAN. The attacker is assumed to have previously deployed a covert network access device in the LAN network, which provides remote access. Implanting this device has been carried out through physical access to one of the hosts within the network, e.g., through an insider who connects such a device to the USB or Ethernet port of an employee's desktop computer; since our focus lies on cyber attacks, this step is considered out of scope of the simulation of Scenario 5.

Through this device, the attacker launches the sniffing tool *bettercap* for ARP spoofing and to capture network traffic. After some time, we assume that a system administrator connected via LAN logs into the video server located within the DMZ. From the collected data, the attacker is able to extract the administrator's authentication token, which they subsequently re-use to make API calls on the video server.

3.4.6 Scenario 6: attack on client

As visualized in Fig. 10, Scenario 6 comprises three main social-engineering-based attack chains targeting a client computer, which lead to initial compromise, persistent access, data collection, and exfiltration. In contrast to

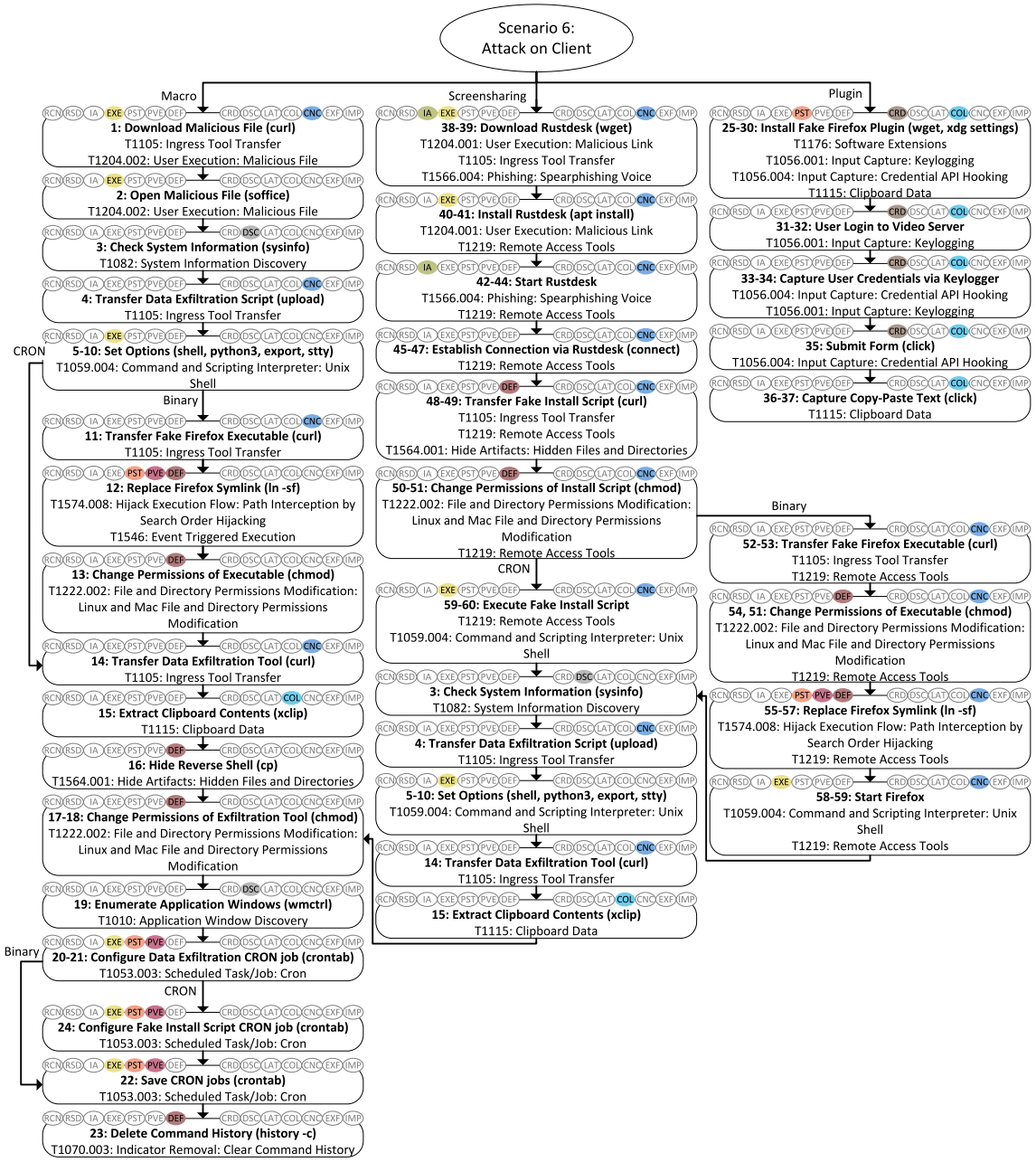


Fig. 10 Attack path of scenario 6: attack on client

previous scenarios, where the attacker primarily relies on tools or exploits of technical vulnerabilities, activities of the human user working on the targeted client computer play a key role for the initial intrusion. Thereby, we assume that the user carries out these activities purposefully but without realizing their malicious nature, for example, through social engineering, where the attacker calls the victim by telephone, pretends to be a technician from the organization's helpdesk, and persuades the user to follow their instructions. Note that in order to avoid that any artifacts from user simulation end up in the log data, we assume that the user remotely accesses their client computer through Virtual Network Computing (VNC). We argue that such remote access software is common in many organizations and does not affect the realism or validity of this attack chain.

Our scenario involves multiple variants; we start our description with the shortest attack chain. (i) *Plugin.* The attacker contacts the user and informs them about the necessary installation of a plugin for the *Firefox* browser as part of a company policy. For the purpose of this scenario, we assume that the user is a web engineer who uses the browser in developer mode, which is necessary for installation. Unknown

to them, the plugin is in fact a keylogger crafted by the attacker, who hosts it on their web server under the unsuspecting domain *dailynews-wire.com*. The attacker talks the user through the process of downloading and installing the malicious plugin as well as configuring and restarting the browser. After ending the call, the attacker only needs to wait for the user to type their username and password when logging into any website. As part of our scenario, we assume that the user navigates to the ZoneMinder website and enter their credentials in the login form, which is captured by the keylogger and transmitted to the attacker. As a final step of this attack chain, the user copies and pastes the contents of the browser's search bar; such clipboard data is also captured and transmitted by the plugin.

The other attack chain in this scenario starts with two variants for initial access. (ii) *Macro*. The attacker informs the victim user about a document containing information on organizational policies. Subsequently, the attacker instructs the victim user to open a terminal and download that document from *dailynews-wire.com*, which appears as a valid domain but is in fact hosted by the attacker. Likewise, the downloaded file seems to be a legitimate text document to the unsuspecting user based on its name and contents. However, the attacker previously implanted a malicious macro into the file that is triggered when opening it with the text editing software *LibreOffice* [80]. When activated, this macro establishes a reverse-shell that provides the attacker with remote access to the system. (iii) *Screensharing*. In this variant, the attacker convinces the user to download and install the legitimate tool *rustdesk* [63] under the disguise of helpdesk software. This tool enables to share and remotely control screens, which supposedly allows the attacker to fix some technical issues on the client's computer. As part of this procedure, the attacker instructs the user on how to configure the tool, shares necessary IP addresses and passwords, and tells them to accept the incoming connection when the attacker activates the same tool on their side. Even though this provides the attacker with remote access on the target system, the remote desktop software makes all actions visible on the user's screen, which means that the attacker's capabilities are limited to non-suspicious activities. In addition, their access is only temporary, because the user needs to manually confirm another connection after closing this session. In order to obtain persistent access, the attacker downloads a reverse-shell from their own typosquatted domain *facebock.com*, stores it as a hidden file, and changes its permissions. We assume that these activities do not trigger any suspicions, as the user fails to notice the misspelled domain and expects the alleged technician to diagnose and resolve issues on their computer.

For the remaining attack chains of *Macro* and *Screensharing* variants we employ similar attack techniques, but slightly vary their order. For brevity, we only explain each step once and refer to Fig. 10 for their position in the attack chain of the respective variant.

Both the reverse-shell established through the malicious macro as well as the remote desktop software only provide temporary access that only lasts until the system is rebooted or the session is terminated. We consider two variants how the attacker obtains persistence on the target system. (i) *Browser Binary*. The attacker downloads a binary for a fake executable for the Firefox browser from their own domain, which opens a reverse-shell before starting the browser as usual. Then, they redirect the symbolic link (symlink) of the actual Firefox browser to point to the fake executable. The attacker thus obtains remote access every time when the user opens the browser, even after the system is restarted. (ii) *CRON*. The attacker creates a CRON job that invokes a previously downloaded script that in turn establishes a remote-shell. Note that this CRON job is configured and started at the end of the attack chain alongside other CRON jobs. From that point on, the script is executed every 5 min even after the system reboots.

Independent of the aforementioned variant, the attacker runs the *sysinfo* command to validate that the remote-shell works as expected and to obtain some information about the system. The attacker's next objective is to extract data from the compromised system. To this end, they transfer and execute *VeilTransfer* [84], a tool that provides several exfiltration techniques and is commonly used in security testing. The attacker uses a script to archive and transfer directories containing potentially sensitive data, in particular, the mail inbox of the Thunderbird application as well as session restore files, key database, and saved credentials of the Firefox browser. At the end of the attack chain, the script is configured as a CRON job that is executed every 10 min.

As an additional exfiltration activity, the attacker extracts clipboard contents and enumerates open application windows as part of collection and discovery activities. As a last step of the attack chain, the attacker clears the command history, which makes it more difficult to trace their activities in case that forensic analyses are carried out once the user becomes aware of the attack.

3.4.7 Scenario 7: docker attack

Figure 11 depicts an attack chain against containerized mail and cloud-storage services. After compromising the cloud service with reused credentials, the attacker escapes the container and gains access to the underlying host.

The scenario starts with an enumeration of the corporate DNS servers identical to the first step of Scenario 1 (cf. Sect. 3.4.1). In this case, however, the scan reveals a mail service that is present in the network. The attacker decides to enumerate user names stored on the server using the *smtp-user-enum* [10] tool, which yields several results. The attacker chooses

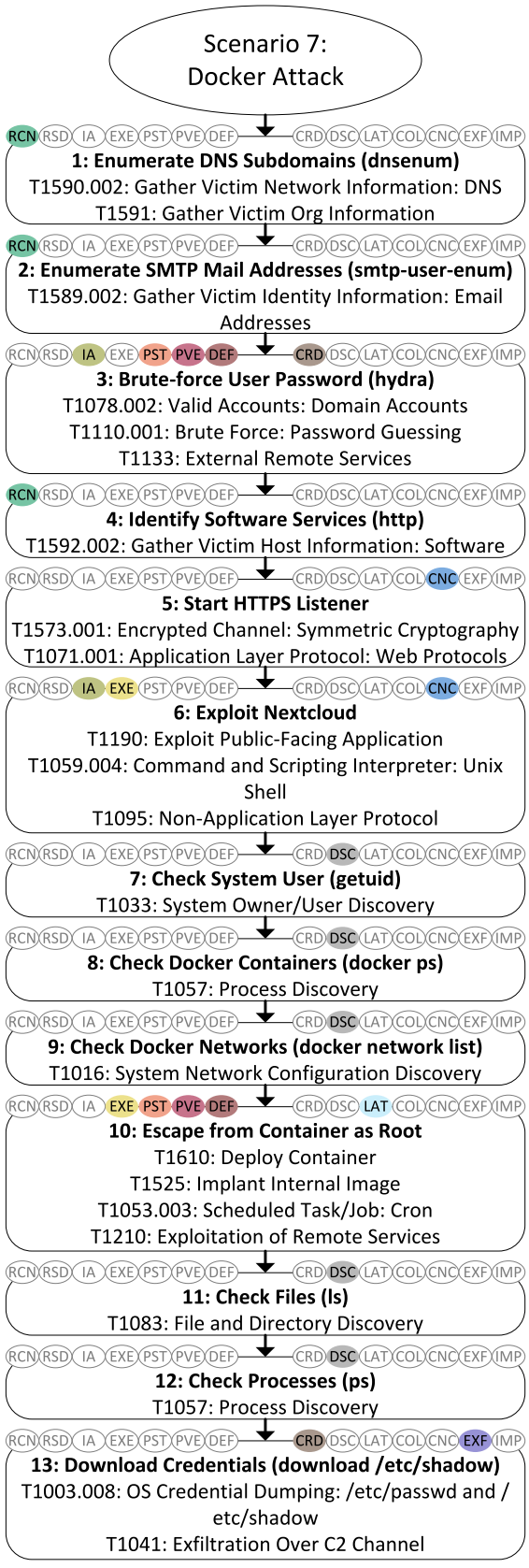


Fig. 11 Attack path of scenario 7: docker attack

user Alice at random and proceeds to brute-force the password of the user's IMAP-account with *hydra* [25]. Note that in Scenario 3, the same tool is used to brute-force an SSH login (cf. Sect. 3.4.3). We also point out that the AttackMate playbook depicted in Fig. 4 illustrates how these initial three steps of the attack chain are implemented for automated execution.

After successfully retrieving the password, the attacker sends a simple web probe to the server and realizes that this host is providing the open-source file sharing platform *Nextcloud* [54] in addition to the mail service. In an attempt to gain access to that system, the attacker executes a Metasploit module that exploits a vulnerable *Nextcloud* application that allows authenticated users to add workflows for command execution [68]. In our scenario, this vulnerable application is present on the platform, which means that the attacker is able to successfully execute the exploit using Alice's mail address and password for authentication.

Through the newly gained shell access, the attacker checks system user information to confirm that they are a non-privileged system user within a containerized environment. At this point, the attacker's primary objective is to escape from the container and gain access on the host running the containers. For the purpose of this scenario, we assume that the Docker API is exposed and accessible from within the container due to weak configurations. This allows the attacker to gather information from the running Docker containers and to enumerate Docker networks. Given this information, the attacker is able to escape the container by temporarily deploying a privileged container through the Docker API, mounting the *etc* directory into that new container, and placing a CRON job that downloads and installs an implant for reverse-shell access into the mounted directory. Once the CRON job triggers, the attacker utilizes the activated implant to access the host system, in particular, to enumerate files and processes as well as to download credential information.

4 Data set analysis

This section provides an analysis of the collected log data sets. We first present an overview of the data and then examine how individual attack steps manifest in the collected logs.

4.1 Data preparation

This section describes the data pre-processing steps applied to isolate attack manifestations. We also provide an overview of these manifestations by visualizing log events as timelines and summarizing the covered attack techniques.

4.1.1 Log event filtering

Analysis of attack manifestations benefits from isolating log events that are direct consequences of attacker activity. As described in Sect. 3.1, we aim to achieve such isolation by operating on an idle infrastructure without user interaction and by waiting 15 min after system boot before initiating attack execution to avoid interference from startup-related processes. Despite these precautions, an initial inspection of the collected data revealed that even idle systems continuously generate a substantial number of log events that inevitably overlap with attack execution. These background events primarily stem from processes that operate independently of user activity, including periodically scheduled system jobs, time synchronization mechanisms, and sporadically occurring system errors such as timeouts.

To mitigate this issue, we apply a filtering approach that removes log events that correspond to normal system behavior with high probability. Specifically, we consider the ten-minute interval preceding the start of AttackMate execution as a baseline period that captures log activity of an idle system; this duration was determined empirically during test runs to capture recurring background events in our environment. We compile log events occurring during this period across all hosts and remove timestamps, counters, and random values from these events using regular expressions. We then filter out all log events from the attack execution phase that are identical to entries in this reference set. We acknowledge that some attack steps may also trigger log events that are identical to those observed during normal system operation and may therefore unintentionally be removed by this process. While the informative value of the contextual occurrence of such events could be useful, we argue that their value for analysis is limited compared to those events that remain due to their syntax or parameters.

Even though this automated filtering approach effectively removes a substantial part of normal background events, we still observe certain events that contain highly variable parameters and are thus difficult to normalize with regular expressions. This particularly affects audit logs, for which we thus apply additional manual filtering based on process identifiers, e.g., to remove events related to intrusion detection activity. To support reproducibility, all manually defined filtering rules are documented and implemented in our open-source code repository. We emphasize that the analyses presented in the following sections only consider the filtered data sets; however, we provide both the filtered and unfiltered data sets in our public repository, allowing researchers to select the version that best fits their use case.

4.2 Timelines

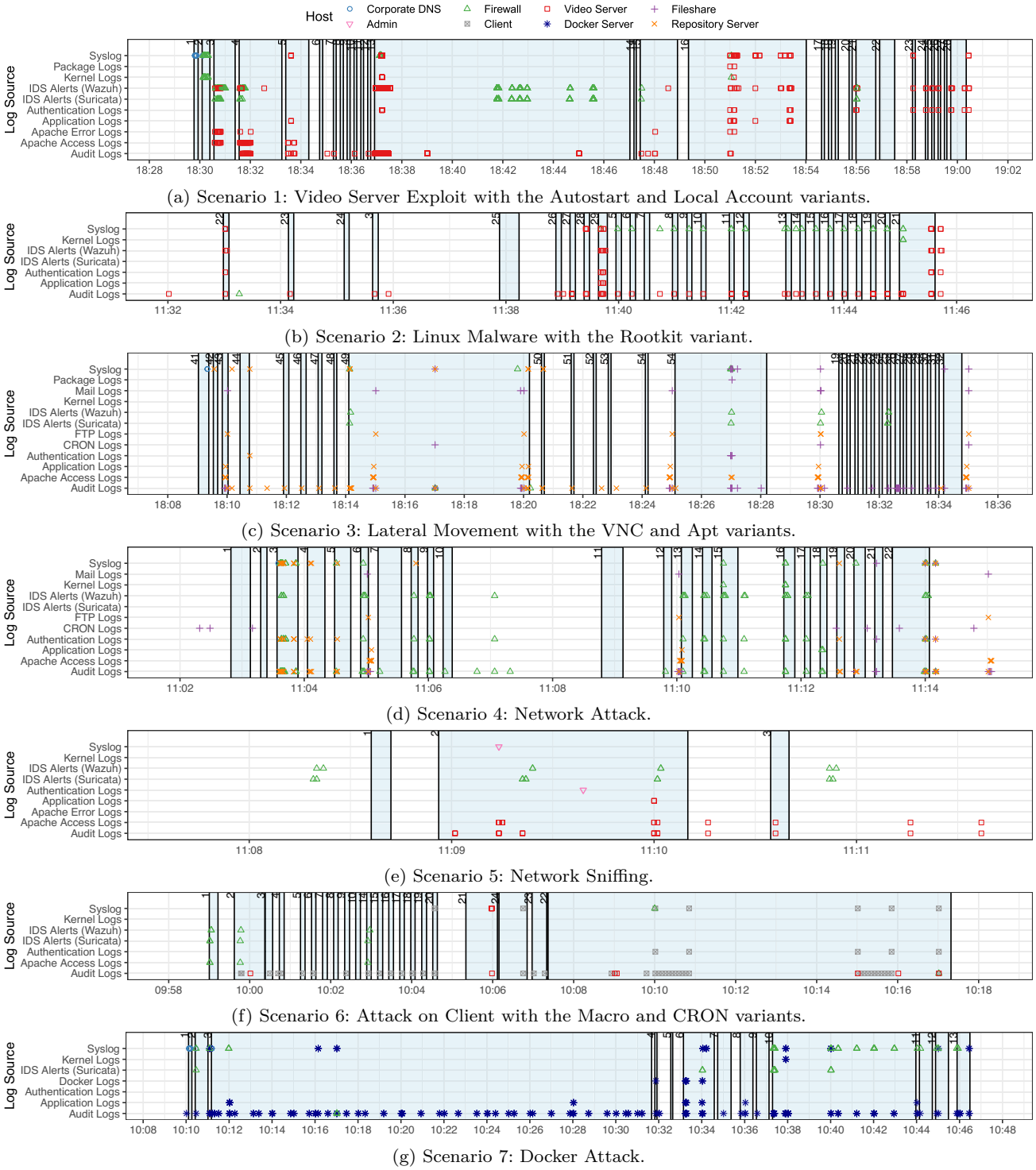
Figure 12 visualizes the attack manifestations of Scenarios 1–7 based on the filtered data sets described in the previous section. For brevity, we display only a single attack path per scenario, using arbitrarily selected variants. Shaded intervals indicate the start and end times of individual attack steps; the numbers above these intervals correspond to the attack step identifiers used in Figs. 5, 6, 7, 8, 9, 10 and 11. Each point in the plots represents the occurrence of a log event or alert, where the vertical axis indicates the source of the event and the color denotes the host on which the event was generated.

The visualizations show that log events accumulate during specific attack steps, with some steps triggering events across multiple log sources or on several hosts. We consider this as an indicator that our filtering procedure successfully removes most events corresponding to normal background activity, while preserving events that are direct consequences of attacker behavior. Furthermore, the figures show that the majority of log events are generated on hosts that are the primary targets of the respective scenarios, such as the video server in Scenario 1 and the client machine in Scenario 6. A small number of events occur outside attack intervals; these correspond either to background activity that was not filtered out or to attack-related events associated with steps that are not explicitly labeled. The figure also illustrates that certain attack steps do not result in any observable log events at all.

4.3 Coverage of attack tactics and techniques

As outlined in Sect. 3.4, the attack scenarios are designed to represent coherent kill chains that collectively cover a broad range of techniques from the MITRE ATT&CK framework [74]. Table 4 summarizes the number of distinct techniques present in each scenario, grouped by their corresponding tactics. Note that techniques associated with multiple tactics are counted separately for each of them. The table shows that all scenarios span multiple tactics, while also revealing that certain scenarios emphasize specific tactics, such as Discovery in Scenario 1 or Impact in Scenario 3.

In addition to the per-scenario breakdown, we also report the total number of distinct techniques covered across all scenarios for each tactic, excluding duplicate techniques that appear in multiple scenarios or are associated with multiple tactics. To contextualize these numbers, we report the corresponding coverage relative to the complete set of techniques defined in the MITRE ATT&CK framework in the final column of the table. As outlined in Sect. 3.1.1, the Resource Development tactic is uncovered because its techniques generally occur outside the monitored infrastructure, while Exfiltration coverage is low because many techniques



(a) Scenario 1: Video Server Exploit with the Autostart and Local Account variants.

(b) Scenario 2: Linux Malware with the Rootkit variant.

(c) Scenario 3: Lateral Movement with the VNC and Apt variants.

(d) Scenario 4: Network Attack.

(e) Scenario 5: Network Sniffing.

(f) Scenario 6: Attack on Client with the Macro and CRON variants.

(g) Scenario 7: Docker Attack.

Fig. 12 Timelines of log event and alert occurrences. Shaded intervals indicate active attack steps

were excluded as difficult to emulate realistically within the test environment.

Overall, the scenarios comprise a total of 81 distinct techniques, corresponding to 37.5% of all techniques defined in the MITRE ATT&CK framework and covering 16 out of the 19 most prevalent techniques mentioned across 667 cyber threat intelligence reports [67]. Several techniques are executed multiple times across different scenarios, using different methods and in varying contexts, which enables a broad analysis of their manifestations in log data. Specifi-

Table 4 Number of attack techniques in scenarios

Tactic	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5	Scenario 6	Scenario 7	Total (Distinct)	MITRE ATT&CK coverage (v18.1)
Reconnaissance	4	0	0	0	0	0	4	5	45.5%
Resource development	0	0	0	0	0	0	0	0	0.0%
Initial access	2	0	2	1	0	1	3	4	36.4%
Execution	2	1	3	0	0	3	3	6	35.3%
Persistence	7	3	4	3	0	4	4	13	56.5%
Privilege escalation	7	3	3	3	0	3	2	9	64.3%
Defense evasion	6	4	4	6	1	4	2	15	31.9%
Credential access	2	1	3	0	3	1	2	7	41.2%
Discovery	16	8	6	1	1	2	4	20	58.8%
Lateral movement	0	1	3	0	1	0	1	5	55.6%
Collection	0	3	2	0	1	2	0	8	47.1%
Command and control	1	3	2	3	0	2	3	6	33.3%
Exfiltration	0	1	0	0	0	0	1	1	11.1%
Impact	0	0	6	0	0	0	0	6	40.0%
Total (Distinct)	35	24	28	10	4	16	22	81	37.5%

cally, 34 distinct techniques are part of at least two scenarios. Furthermore, some techniques are realized through multiple sub-techniques, which are labeled accordingly. When accounting these labels, the scenarios cover a total of 97 out of 475 distinct sub-techniques specified in MITRE ATT&CK.

4.4 Cyber attack manifestations

This section analyzes how attack techniques manifest in system log data.

4.4.1 Observability of attack commands

Both manual log analysis by experts as well as LLM-based log interpretation rely heavily on the expressiveness of log event contents. Ideally, every action carried out by attackers generates log events that contain sufficiently granular information to make the underlying activity straightforward to understand. In practice, however, logs often capture only indirect consequences of actions rather than the actions themselves, or lack essential contextual information. To better understand how attacks manifest in log data, we investigate which attack techniques generate log events that contain explicit indicators of attacker activity.

To this end, we extract the commands executed by our attack emulation tool for each attack step and search the collected log data for occurrences of these commands. Specifically, we split each command into individual tokens on spaces and remove common stop-words (e.g., “sudo”, “id”) that yield too many matches in unrelated events. We consider an attack step to contain an explicit command indicator if at least one of the remaining tokens appears in any log events captured during the time interval where that attack step was captured.

Figure 13 summarizes the results of this experiment. Overall, a substantial fraction of attack steps generate log events that contain information about the executed command. Out of the 243 attack steps in our data set (cf. Sect. 3.4), 115 (47.3%) leave such indicators in audit logs. This result is largely explained by the fact that many attack chains involve direct command execution on compromised systems, for example as part of information gathering. Since audit logs record executed commands, they constitute a particularly rich source of information for log interpretation. Figure 14a shows sample audit log events capturing the command “cat /etc/shadow”, which the attacker uses to dump credential hashes. Note that in some cases, audit logs encode command information in hexadecimal form; in the provided example, decoding the “proctitle” field also reveals the executed command “cat /etc/shadow”.

Other log sources, including syslog, access logs, authentication logs, etc., also contribute relevant information. In addition to audit logs, 42 attack steps (17.3%) leave explicit command indicators in other log sources, while 5 attack steps (2.1%) generate such indicators exclusively outside of audit logs. The latter primarily involves remote tools that are not executed directly on the target system. For example, the access logs shown in Fig. 15 correspond to scanning activity and contain the keyword “nikto”, identifying the scanner used by the attacker from outside the network. At the same

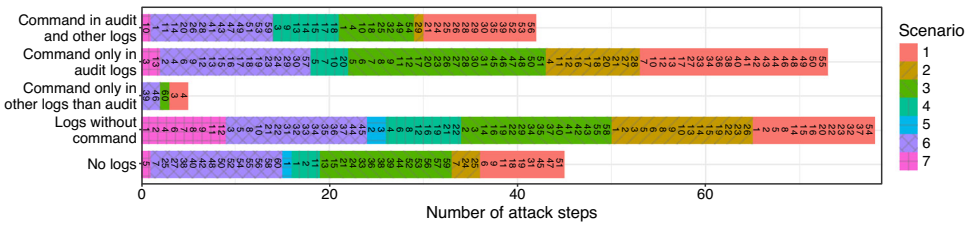


Fig. 13 Observability of attacks and commands executed by the attacker in log data

time, our analysis shows that 78 attack steps (32.1%) generate log events that do not directly reference the executed command, and 45 attack steps (18.5%) that do not involve any log events at all after filtering.

Importantly, the presence of command strings in log data does not imply that an attack step is straightforward to interpret. First, identifying relevant commands becomes challenging when they are concealed by large volumes of log events with diverse parameters. Second, correct interpretation often depends on contextual information. For instance, audit logs generally record executions of commands such as "chmod", which are useful to make attack scripts executable (e.g., Scenario 6, Step 13); however, the same command may also be applied on benign files as part of routine administrative activity. Distinguishing benign from malicious use thus requires additional context, such as affected files and directories, the involved user, the temporal context of the action, etc.

A key finding of our analysis is that attack manifestations depend not only on which command is executed, but also on how it is executed. Figure 14 illustrates this effect using three attack steps from different scenarios that all execute the same command: "cat /etc/shadow". In Scenario 3, the attacker executes the command via an interactive VNC session, resulting in audit logs that clearly expose both the command and its arguments (cf. Fig. 14a). In Scenario 7, the same command is executed through an implant, and the corresponding logs (cf. Fig. 14b) reveal only the accessed file path while obscuring the command itself. In Scenario 1, where the attacker interacts with the system through a reverse-shell, the command appears in authentication logs (cf. Fig. 14c). Since even identical commands may surface in different log sources and with varying levels of granularity depending on the execution method, we conclude that no consistent mapping between attack techniques and specific log manifestations can be established.

4.4.2 Log event frequencies

Aside from the contents of individual log messages, analysts frequently monitor the frequency of log events, for example, through dashboards that aggregate event counts over fixed time intervals. Changes in event frequencies are commonly used as basic indicators of anomalies in system behavior, because significant deviations such as spikes that stand out from long-term baselines may point to system issues or malicious activity [26, 39].

Several attack techniques included in our data set generate large volumes of log events. It is difficult to determine, however, which of these techniques would trigger suspicion in operational environments, because this kind of detection depends on the individual event count baselines. Since our data set does not include simulations of benign workload behavior, we only compare the frequencies of log events generated by different attack steps with each other. Figure 16 visualizes the number of log events produced per attack step across log sources. Note that the vertical axis uses a logarithmic scale. As shown in the figure, audit logs account for the largest share of events. The main reason for this is that a large number of attack steps directly interact with systems and thus trigger audit logging, causing anything from a few to several thousands of events. Thereby, the system vulnerability scan performed in Step 13 of Scenario 1 produces the highest volume of audit logs, generating more than 40,000 events.

In contrast, most other log sources typically do not generate more than 100 events per attack step, often substantially fewer. Notable exceptions to this tendency are attack steps involving network-based scanning activities. For example, the DNS enumeration in Step 1 of Scenario 1 results in a high number of syslog entries on the DNS server, the software installation in Step 41 of Scenario 6 creates many logs from package management services, and the network vulnerability scan in Step 3 of Scenario 1 produces a large volume of access logs. An excerpt of the latter is shown in Fig. 15, which depicts a rapid succession of requests corresponding to entries from a wordlist. Similar log characteristics can be observed for the brute-force login executed in Step 1 of Scenario 3. Figure 17 shows an excerpt of the corresponding logs that document a series of failed authentication events taking place within a few seconds.

4.4.3 System performance metrics

Log collection frameworks such as collectd capture system performance metrics over time. The collected measurements provide insights into system utilization at given points in time and are particularly useful for correlating system activi-

```
type=SYSCALL msg=audit(1765563116.496:15535): arch=c000003e syscall=59 success=yes exit=0 a0=5583f2d27de0 a1
    =5583f2e33fe0 a2=5583f2e2ff80 a3=8 items=2 ppid=4269 pid=4289 auid=4294967295 uid=0 gid=0 euid=0 suid=0
    fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts2 ses=4294967295 comm="cat" exe="/usr/bin/cat" subj=unconfined key="
    T1078_Valid_Accounts"
type=EXECVE msg=audit(1765563116.496:15535): argc=2 a0="cat" a1="/etc/shadow"
type=PATH msg=audit(1765563116.496:15535): item=0 name="/usr/bin/cat" inode=1568 dev=fc:01 mode=0100755 ouid
    =0 ogid=0 rdev=00:00 nametype=NORMAL cap_fp=0 cap_fi=0 cap_fe=0 cap_fver=0 cap_frootid=0
type=PATH msg=audit(1765563116.496:15535): item=1 name="/lib64/ld-linux-x86-64.so.2" inode=79850 dev=fc:01
    mode=0100755 ouid=0 ogid=0 rdev=00:00 nametype=NORMAL cap_fp=0 cap_fi=0 cap_fe=0 cap_fver=0 cap_frootid=0
type=PROCTITLE msg=audit(1765563116.496:15535): proctitle=636174002F6574632F736861646F77
```

(a) Command visible as clear text and hexadecimal-encoded text in audit logs of Scenario 3, Step 45.

```
type=SYSCALL msg=audit(1768992356.109:5653): arch=c000003e syscall=257 success=yes exit=7 a0=ffffffffffffff9c
    a1=c00046ec50 a2=80000 a3=0 items=1 ppid=3689 pid=3717 auid=0 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0
    sgid=0 fsgid=0 tty=(none) ses=31 comm="tool" exe="/opt/tool" subj=unconfined key="T1087_Account_Discovery
    "
type=PATH msg=audit(1768992356.109:5653): item=0 name="/etc/shadow" inode=5154 dev=fd:01 mode=0100640 ouid=0
    ogid=42 rdev=00:00 nametype=NORMAL cap_fp=0 cap_fi=0 cap_fe=0 cap_fver=0 cap_frootid=0
type=PROCTITLE msg=audit(1768992356.109:5653): proctitle="/opt/tool"
```

(b) Parts of command visible in audit logs of Scenario 7, Step 13.

```
Sep 22 18:58:45 videoserver sudo: webadmin : PWD=/home/webadmin ; USER=root ; COMMAND=/usr/bin/cat /etc/
    shadow
Sep 22 18:58:45 videoserver sudo: pam_unix(sudo:session): session opened for user root(uid=0) by (uid=1003)
Sep 22 18:58:45 videoserver sudo: pam_unix(sudo:session): session closed for user root
```

(c) Command visible in authentication logs of Scenario 1, Step 24.

Fig. 14 Observability of the attack command for credential dumping (T1003.008) across scenarios

```
192.42.1.174 - - [22/Sep/2025:18:30:44 +0000] "GET /old/ HTTP/1.1" 404 396 "-" "Mozilla/5.00 (Nikto/2.1.5)"
192.42.1.174 - - [22/Sep/2025:18:30:44 +0000] "GET /oracle HTTP/1.1" 404 396 "-" "Mozilla/5.00 (Nikto/2.1.5)"
192.42.1.174 - - [22/Sep/2025:18:30:44 +0000] "GET /oradata/ HTTP/1.1" 404 396 "-" "Mozilla/5.00 (Nikto/2.1.5)
    "
192.42.1.174 - - [22/Sep/2025:18:30:44 +0000] "GET /order/ HTTP/1.1" 404 396 "-" "Mozilla/5.00 (Nikto/2.1.5)"
192.42.1.174 - - [22/Sep/2025:18:30:44 +0000] "GET /orders/ HTTP/1.1" 404 396 "-" "Mozilla/5.00 (Nikto/2.1.5)"
192.42.1.174 - - [22/Sep/2025:18:30:44 +0000] "GET /orders/checks.txt HTTP/1.1" 404 396 "-" "Mozilla/5.00 (
    Nikto/2.1.5)"
```

Fig. 15 Excerpt of the log events generated as a consequence of scanning activities from Scenario 1, Step 3

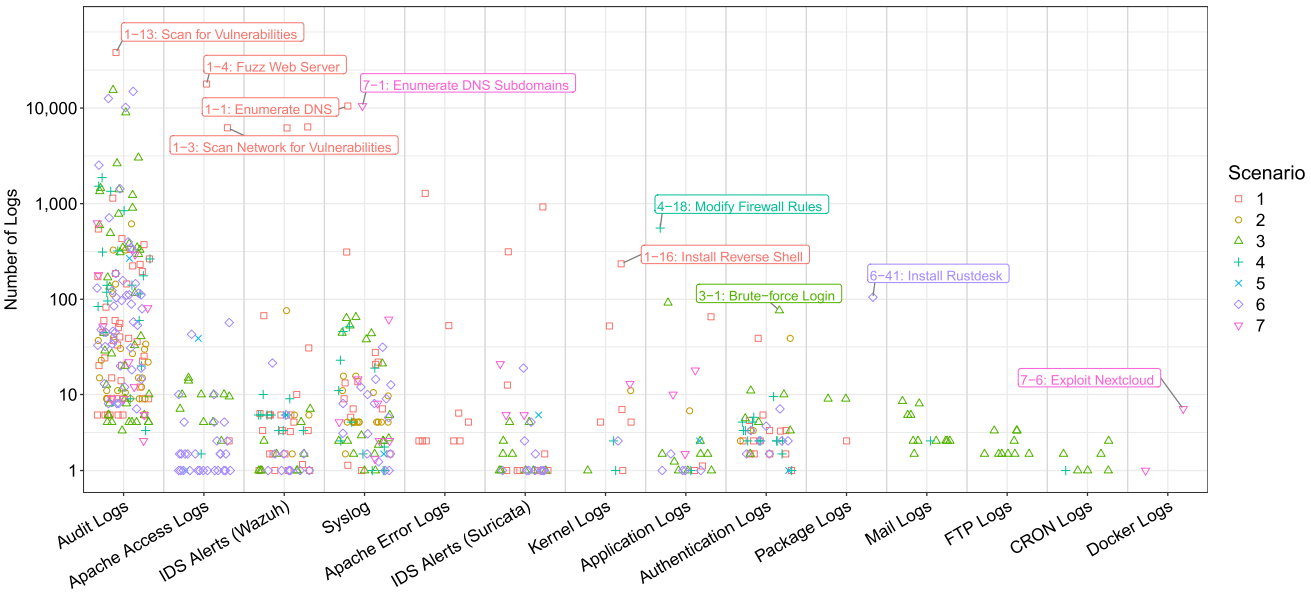


Fig. 16 Log event frequencies observed in different log sources. Labels mark selected attack steps associated with particularly high numbers of log events

```
Dec 11 14:28:03 puppet sshd[3089]: Invalid user admin from 192.42.1.174 port 33006
Dec 11 14:28:03 puppet sshd[3089]: Received disconnect from 192.42.1.174 port 33006:11: Bye Bye [preauth]
Dec 11 14:28:03 puppet sshd[3095]: pam_unix(sshd:auth): check pass; user unknown
Dec 11 14:28:03 puppet sshd[3095]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh
    ruser= rhost=192.42.1.174
Dec 11 14:28:05 puppet sshd[3095]: Failed password for invalid user admin from 192.42.1.174 port 56768 ssh2
```

Fig. 17 Failed authentication events generated as a consequence of a brute-force login from Scenario 3, Step 1

```
[2025-09-22 18:37:00] [info] write_log values:
videoserver_novalocal.load.load.shortterm 0.1
[2025-09-22 18:37:10] [info] write_log values:
videoserver_novalocal.load.load.shortterm 0.24
[2025-09-22 18:37:20] [info] write_log values:
videoserver_novalocal.load.load.shortterm 1.12
[2025-09-22 18:37:30] [info] write_log values:
videoserver_novalocal.load.load.shortterm 1.71
[2025-09-22 18:37:40] [info] write_log values:
videoserver_novalocal.load.load.shortterm 1.44
```

Fig. 18 System metric captured with collectd

ties with changes in system performance, for example, when programs with high computational workload are executed. Common categories of metrics include CPU, file system, disk, entropy, network interfaces, interrupt counters, load, memory, processes, and swap usage; each category may comprise multiple individual measurements. Figure 18 shows an excerpt of collectd logs capturing short-term system load in intervals of ten seconds. The displayed measurements indicate a noticeable increase in system load, which is caused by the system vulnerability scan (Scenario 1, Step 13).

Figure 19 visualizes the evolution of system performance metrics captured during the entire simulation period of Scenario 1 for the selected variants Autostart and Local Account (cf. Sect. 3.4.1). The displayed metrics include system load, CPU utilization of user-space processes, the number of disk read operations, total memory usage, the number of received network packets, and the number of running processes. Note that for visualization purposes, all metrics are min-max scaled to a common range. The figure reveals that certain attack steps have a measurable impact on system performance. Corresponding to the logs displayed in Fig. 18, the system vulnerability scan executed in Step 13 leads to a sharp increase in CPU utilization, disk read activity, and system load. These results demonstrate that attack steps do not only manifest through generation of log events but also impact measurements in continuously generated system performance metrics.

4.4.4 Intrusion detection alerts

Certain types of log events are considered suspicious because they are commonly produced as consequences of malicious or otherwise undesired system activities. Similarly, specific behavioral patterns observable in network traffic can indicate potential attacks. Intrusion Detection Systems (IDS) analyze system logs and network traffic for matches against predefined detection rules and signatures, triggering alerts when such patterns are observed. As described in Sect. 3.1.3, our experimental setup includes the host-based IDS Wazuh and the network-based IDS Suricata, both configured to use their respective publicly available default rule sets.

Figures 20 and 21 illustrate sample alerts generated by these systems. Figure 20 shows alerts triggered by Wazuh during Step 15 of Scenario 4, in which the attacker installs a port-knocking tool. The first alert is a high-severity alert (level 10) indicating that a network interface has been switched into promiscuous mode, which is typically associated with network monitoring and therefore considered security-critical. The second alert has a substantially lower severity (level 3) and records the execution of a privileged command. The main reason for the low severity is that these commands are common in benign administrative workflows. Figure 21 presents two alerts generated by Suricata during Step 25 of Scenario 3, in which the attacker downloads ransomware. The first alert has medium severity (priority 2) and flags an HTTP request to an IP address, which is in our case controlled by the attacker. The second alert has the highest severity level (priority 1) and is triggered because an executable file is downloaded over unencrypted HTTP. Note that Suricata and Wazuh use different severity scales, which necessitates normalization for comparative analysis.

We analyze all IDS alerts triggered during the execution of our attack scenarios. Figure 22 summarizes the attack steps that result in alerts and groups them by severity. To enable comparability across IDS, we map Wazuh's severity levels onto the three-point scale used by Suricata as follows: levels 4-6 correspond to Low, levels 7–9 to Medium, and level 10 and above to High. Wazuh alerts with severity of level 3 or lower are treated as Informational, as they do not necessarily indicate malicious activity. Note that single attack steps may trigger multiple alerts across different severity levels (cf. Figs. 20 and 21) and from both IDSs at the same time. For example, the scanning activities in Steps 3 and 4 of Scenario 1 generate alerts from both Wazuh and Suricata, spanning all severity categories. Overall, the results show that Suricata triggers alerts for a larger number of attack steps in the Low-High severity ranges, whereas Wazuh predominantly generates informational alerts across many steps.

Our analysis shows that out of the 198 attack steps that produce log data, only 43 trigger alerts with a severity level

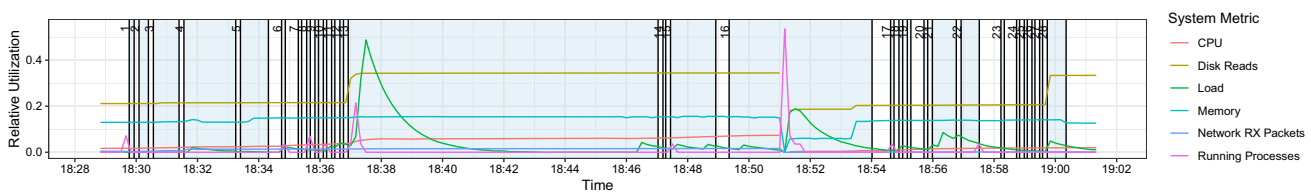


Fig. 19 Timelines of system performance metrics collected from the video server in Scenario 1

of Low-High in at least one of the two IDS. An additional 20 attack steps trigger informational alerts in Wazuh. Conversely, 155 attack steps remain undetected by either IDS when relying on default detection signatures. These results highlight the limitations of signature-based intrusion detection in capturing a wide range of attacker activities and underscore the importance of complementary analysis approaches, such as methods for anomaly-based detection or LLM-based log interpretation.

5 LLM-based interpretation

This section presents an illustrative evaluation of our data set in the context of log interpretation using LLMs.

5.1 Experimental setup

In this section, we summarize the setup of our experiment.

5.1.1 Task and models

We task LLMs with interpreting log data in a zero-shot setting, i.e., without providing any training samples. The goal of this experiment is to approximate a realistic usage scenario in which security analysts copy and paste log excerpts into an LLM to obtain a fast and automated first assessment that complements manual analysis [75]. We evaluate GPT-5.5,[7] GPT-5.2,[8] Llama-4,[9] Qwen3,[10] and Ministral.[11] We select these models to cover a representative spectrum of contemporary LLMs. The selection includes both proprietary (GPT) and open-weight (Llama, Qwen, Ministral) models, multiple model families developed by different organizations, and a range of model sizes. GPT models have also been successfully applied to log analysis in prior work [44, 65]. Thereby, GPT-5.2 predates the release of our dataset, reducing the likelihood of data leakage through pretraining, while GPT-5.5 represents the most capable GPT model available at the time of carrying out the experiments. All models are accessed through their respective APIs. We configure the temperature to zero for all models to minimize hallucinations.

5.1.2 Prompting

For each of the 198 attack steps that generate log data, we submit one prompt to each LLM. Thereby, each prompt starts with a task description instructing the model to (i) classify the provided log data according to the MITRE ATT&CK framework and return the Top 10 most relevant techniques in descending order, (ii) estimate whether the logs correspond to malicious or benign behavior on a 7-point Likert scale, and (iii) provide a short explanation on the reasoning behind the assessment.

Following this task description, we include log data collected from all hosts and log sources involved in the respective attack step. To prevent labels embedded in the logs from influencing the LLM's predictions, we use regular expressions to remove any references to MITRE ATT&CK tactics and techniques. Furthermore, to avoid excessive prompt lengths, we limit the number of logs per host-source pair at ten lines. These lines are sampled at random positions within the corresponding log file. We repeat this process five times per attack step with new random samples of log lines in order to mitigate the effects of sampling randomness and to assess the consistency of resulting classifications. We acknowledge that this sampling strategy may exclude particularly informative events and discuss this limitation in Sect. 6.2.

Figure 23 shows an exemplary prompt, including both the instruction preamble and selected access and audit logs from the video server during the ZoneMinder exploit (Scenario 1, Step 5). The logs depict the request to the web server that transmits the malicious payload, a part of the subsequent execution of the payload on the system itself, as well as a warning message about a missing event from the ZoneMinder application. Note that we simplify some logs and remove less relevant log sources for brevity.

5.2 Results

This section presents the results obtained from our experiment.


```
{
  "rule": {
    "level": 10,
    "description": "Auditd: Device enables
        promiscuous mode.",
    "id": "80710"
  },
  "full_log": "type=ANOM_PROMISCUOUS msg=audit
      (1765797045.372:6848): dev=ens3 prom=256
      old_prom=0 auid=4294967295 uid=0 gid=0 ses
      =4294967295",
  "location": "/var/log/audit/audit.log"
}
{
  "rule": {
    "level": 3,
    "description": "Successful sudo to ROOT executed.
        "
    "id": "5402"
  },
  "full_log": "Dec 15 11:10:45 inetfw sudo: root :
      TTY=pts/1 ; PWD=/usr/bin ; USER=root ; COMMAND
      =/usr/bin/systemctl start auditf.service",
  "location": "/var/log/auth.log"
}
```


Fig. 20 Alerts triggered by Wazuh IDS when installing the port knocking tool in Scenario 4, Step 15

[7] openai/gpt-5.5-20260423.

[8] openai/gpt-5.2-20251211.

[9] meta-llama/llama-4-maverick-17b-128e-instruct.

[10] qwen/qwen3-32b-04-28.

[11] mistralai/ministral-3b-2512.

```
12/11/2025-14:42:48.133080  [**] [1:2034567:1] ET HUNTING curl User-Agent to Dotted Quad [**] [Classification:
    Potentially Bad Traffic] [Priority: 2] {TCP} 192.168.100.23:49346 -> 192.42.1.174:80
12/11/2025-14:42:48.133803  [**] [1:2019240:14] ET POLICY Executable and linking format (ELF) file download
    Over HTTP [**] [Classification: Potential Corporate Privacy Violation] [Priority: 1] {TCP} 192.42.1.174:80
    -> 192.168.100.23:49346
```

Fig. 21 Alerts triggered by Suricata IDS when downloading ransomware in course of Scenario 3, Step 25

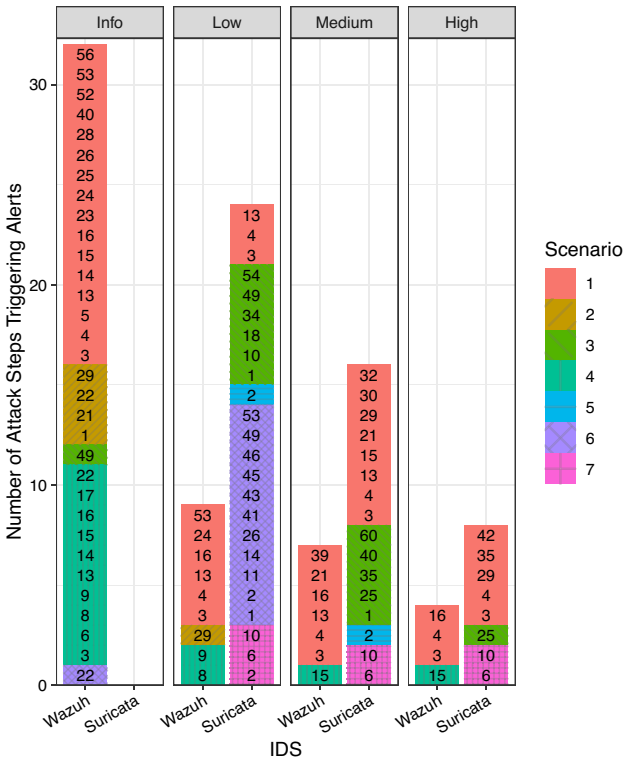


Fig. 22 Severities of intrusion detection alerts across scenarios and attack steps

5.2.1 Qualitative examples

Figure 24 showcases three correct and two incorrect interpretations of attack steps generated by GPT-5.5. Figure 24a shows the response to the prompt in Fig. 23. Based on the explanation generated by the LLM, it becomes evident that the model correctly connects the web request observed in the access logs with the command executed by the web application process recorded in the audit logs, and identifies this behavior as unusual. This form of temporal correlation and multi-source reasoning resembles the analytical process expected from a human security analyst. The ground truth labels of the corresponding attack step, namely *Exploitation of Public-Facing Application (T1190)* as well as *Command and Scripting Interpreter (T1059)*, appear in all of the first four positions of the top predictions, indicating a strong semantic understanding of both the meaning of the log content and the associated attack techniques.

The response in Fig. 24b demonstrates the interpretation of credential dumping; relevant logs are depicted in Fig. 14c . The response shows that the correct technique and sub-technique (*T1003.008*) appear at the first position in the ranked list of predictions. Again, the explanation of the underlying cause of the logs and their relevance to specific malicious activities are on point.

Figure 24c shows the interpretation of vulnerability scanning, with log excerpts shown in Fig. 15. In this case, the LLM correctly identifies the scanning tool via the user-agent string as well as the high number of requests, thus leveraging both event content (cf. Sect. 4.4.1) and frequency (cf. Sect.

```
You are a MITRE ATT&CK TTP classification expert. Your task is to classify the following system log data. You
    are provided with samples from one or more hosts and one or more log sources that are captured during
    execution of one specific MITRE ATT&CK technique. Always output a valid JSON object with the following
    fields:
- "techniques": A list of top 10 ATT&CK techniques that are most likely related to the sample logs, sorted in
    descending order. Only print the ID of the techniques without any other descriptions.
- "confidence": An estimate for the certainty that the logs indicate an actual attack rather than normal
    system or user activity. Provide one of the following estimates: "Certain: Attack", "Almost Certain:
    Attack", "Somewhat Certain: Attack", "Neutral", "Somewhat Certain: Normal", "Almost Certain: Normal", "
    Certain: Normal"
- "explanation": A brief explanation (1-2 sentences) why you think that the samples correspond to attacks or
    normal behavior, e.g., by pointing to specific artifacts or properties of the logs.

videoserver:
 logs/access.log.1:
  192.42.1.174 - - [24/Sep/2025:11:31:05 +0000] "GET /zm/index.php HTTP/1.1" 200 8978 "-" "Mozilla/5.0 (
      Windows NT 10.0; Win64; x64; rv:133.0) Gecko/20100101 Firefox/133.0"
  192.42.1.174 - - [24/Sep/2025:11:31:05 +0000] "POST /zm/index.php HTTP/1.1" 302 756 "-" "Mozilla/5.0 (
      Windows NT 10.0; Win64; x64; rv:133.0) Gecko/20100101 Firefox/133.0"
 logs/log/zm/web_php.log:
  9/24/25, 11:31:12 AM UTC.759462 web_php[2639].WAR [192.42.1.174] [No event returned from TriggerOn] at /usr/
      share/zoneminder/www/includes/Monitor.php line 678
 logs/log/audit/audit.log:
  type=SYSCALL msg=audit(1758713480.172:4453): arch=c000003e syscall=59 success=yes exit=0 a0=55df1de3c000 a1
      =55df1de3bf88 a2=55df1de3bfa0 a3=8 items=4 ppid=2688 pid=2689 auid=4294967295 uid=33 gid=33 euid=33 suid
      =33 fsuid=33 egid=33 sgid=33 fsgid=33 tty=(none) ses=4294967295 comm="which" exe="/usr/bin/dash" subj==
      unconfined ARCH=x86_64 SYSCALL=execve AUID="unset" UID="www-data" GID="www-data" EUID="www-data" SUID="
      www-data" FSUID="www-data" EGID="www-data" SGID="www-data" FSGID="www-data"
  type=EXECVE msg=audit(1758713480.172:4453): argc=3 a0="/bin/sh" a1="/usr/bin/which" a2="python"
```

Fig. 23 Simplified excerpt of an LLM prompt showing a task description and logs from Scenario 1, Step 5

```
"techniques": ["T1190", "T1059.004", "T1059", "T1059.006", "T1105", "T1082", "T1046", "T1053", "T1210", "T1505.003"],
"confidence": "Almost Certain: Attack",
"explanation": "The web server logs show ZoneMinder access immediately followed by audit events where the web service user www-data executes /bin/sh to run /usr/bin/which python, indicating command execution through the public-facing web application. This pattern is consistent with exploitation of a web application vulnerability leading to shell command execution rather than normal ZoneMinder activity."
```


(a) ZoneMinder exploit in Scenario 1, Step 5.


```
"techniques": ["T1003.008", "T1003", "T1078.003", "T1548.003", "T1552.001", "T1087.001", "T1069.001", "T1059.004", "T1202", "T1033"],
"confidence": "Almost Certain: Attack",
"explanation": "The logs show user debian using sudo to run /usr/bin/cat /etc/shadow, with audit records confirming successful open of /etc/shadow as root. Accessing the Linux shadow password file is a strong indicator of OS credential dumping rather than routine administration."
```


(b) Credential dumping in Scenario 1, Step 24.


```
"techniques": ["T1595.002", "T1190", "T1083", "T1595", "T1592.002", "T1210", "T1046", "T1590.005", "T1059.007", "T1005"],
"confidence": "Certain: Attack",
"explanation": "The logs show an automated Nikto web vulnerability scan from 192.42.1.174 with many 404 probes for CGI/PHP paths, directory traversal attempts targeting /etc/passwd and boot.ini, and Suricata alerts for XSS and protected file access. This is strongly indicative of active web application reconnaissance/vulnerability scanning rather than normal user activity."
```


(c) Vulnerability scanning in Scenario 1, Step 3.


```
"techniques": ["T1021.004", "T1078", "T1110", "T1133", "T1036", "T1005", "T1059.004", "T1087", "T1018", "T1070.002"],
"confidence": "Somewhat Certain: Normal",
"explanation": "The logs primarily show a successful SSH login for user john from 192.42.1.174 and normal sshd activity such as reading hosts.allow/deny and updating wtmp/lastlog. There are no clear malicious artifacts such as brute-force failures, anomalous commands, or tampering, though remote SSH access maps to ATT&CK techniques if used by an attacker."
```


(d) SSH Login in Scenario 4, Step 5.


```
"techniques": ["T1105", "T1071.001", "T1204.002", "T1583.001", "T1566.002", "T1036.005", "T1189", "T1059.004", "T1041", "T1090"],
"confidence": "Almost Certain: Attack",
"explanation": "Suricata reports a curl User-Agent connecting to an external dotted-quad host on port 8888 followed by an ELF file download, and DNS logs show repeated lookups for the suspicious typosquatted domain faaacebook.com resolving to that host. These artifacts strongly indicate retrieval of a Linux payload/tool over the network rather than normal administrative Docker/network activity."
```


(e) Container escape in Scenario 7, Step 10.

Fig. 24 Samples of LLM-based classification and interpretation of logs from attack steps

4.4.2). Once more, the top prediction matches the ground truth sub-technique *Active Scanning: Vulnerability Scanning (T1595.002)* and is even assigned the highest confidence level (*Certain: Attack*).

Figure 24d depicts the LLM's interpretation of the attacker's SSH access. As visible in the output, the ground-truth technique, *Valid Accounts (T1078)*, is ranked second in the list of predictions, immediately after the closely related *Remote Services: SSH (T1021.004)*. However, despite identifying the behavior correctly, the model incorrectly assigns the confidence category *Somewhat Certain: Normal*. This error is understandable given that the attacker misuses a legitimate user account, which makes the activity difficult to distinguish from benign access when considered in isolation. Providing additional context from surrounding events, such as malicious actions performed by the same user after login, could help the model interpret such activities more accurately.

Finally, Fig. 24e illustrates a misclassification involving a container escape attack. The model correctly identifies the logs as suspicious and even links them to Docker activity and communication with a suspicious domain, but fails to infer the underlying attack technique and key purpose of this step. None of the ground-truth techniques, including *Deploy Container (T1610)*, appears among the Top 10 predictions, which instead contain more generic techniques related to tool transfer (*T1105*) and execution (*T1204.002*). A possible explanation for this incorrect interpretation is that the presence of Suricata alerts that emphasize indicators such as *curl to dotted quad* and *ELF-file download*, which may steer the model toward these more generic interpretations. In addition, the sampled log excerpt may not provide sufficient context to reveal the full activity. For example, the complete Docker audit log and syslog file contain 625 and 31 lines respectively, whereas our sampling strategy selects only ten lines from each log source.

5.2.2 Quantitative evaluation

We evaluate classification performance by comparing the predicted techniques with the ground truth labels across all five runs of the 198 attack steps. Thereby, we only consider matches at the technique level since the model was

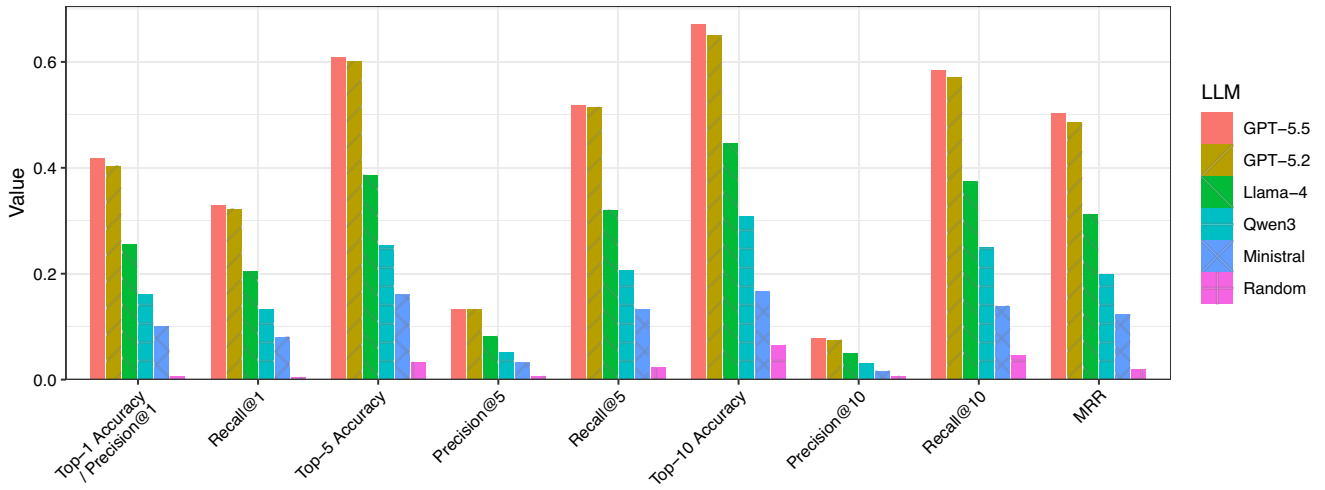


Fig. 25 Accuracy of LLM-based attack technique classification

not instructed to return sub-technique identifiers. Any duplicated techniques are removed after mapping sub-techniques to their corresponding techniques while preserving the original prediction order.

Metrics. Our quantitative evaluation relies on the following metrics that are averaged over all attack steps. *Top-k Accuracy* measures whether at least one correct technique from the list of ground truth labels appears among the first k predictions. Given that ground truth labels often involve multiple techniques and matching of several techniques indicates a better classification, we also compute precision and recall. *Precision@k* measures the fraction of the first k predictions that are correct. *Recall@k* measures the fraction of ground-truth techniques that appear in the first k predictions. As an additional metric that is independent from any fixed k, we compute the *Mean Reciprocal Rank (MRR)* that considers the position of correct techniques in the list of predictions and is highest when correct techniques are ranked first. Specifically, this metric computes an average of the inverse rank of the first correct prediction within the Top 10. Across all of these metrics, higher values indicate a better classification.

Aside from the correctness of results, we also report on the stability observed over the sample runs. *Same-Top-1* measures the fraction of attack steps that yield the same Top 1 technique prediction across all samples. *Pair-Top-1* computes the fraction of matching pairs of Top 1 technique predictions over all possible combinations and averages the scores for all attack steps. Other than *Same-Top-1*, this score thus also considers partial matches within the sample runs. *Fleiss-Kappa* measures the reliability of Top 1 technique predictions beyond what would be expected by chance [15]. Note that for each of these three stability metrics, higher scores indicate a more stable outcome of the classification process. Moreover, these metrics are independent of ground truth labels, i.e., both correct and incorrect predictions can be stable if they are consistent across runs. As a complementary measure, we therefore also compute the *Rank-Stdev*, defined as the standard deviation of the rank of the first prediction matching any ground-truth technique across runs. Runs in which no ground-truth technique appears among the Top 10 predictions are excluded, since assigning them an identical fallback rank would artificially reduce the observed variation. While the other stability metrics are bounded within [0, 1], *Rank-Stdev* ranges from 0 to 4.5 (the maximum possible population standard deviation for values in [1, 10]), with lower values indicating more stable rankings.

Results. Figure 25 summarizes the accuracy of the considered LLMs for assigning attack techniques to log samples. To put the classification performance of the LLMs into perspective, we include a random classifier that draws predictions uniformly at random and without replacement from the set of all 216 available MITRE ATT&CK techniques. Overall, all models identify correct techniques above the random baseline, but their performance differs substantially. The two GPT models achieve the strongest results across all accuracy measures. GPT-5.5 performs best overall, ranking at least one correct technique first for 41.8% of attack steps and within the Top 10 for 67.2%, slightly outperforming GPT-5.2. Ministral, the smallest model considered, achieves the lowest accuracy among the LLMs, but still remains clearly above the random baseline. This overall ranking is consistent across the remaining metrics related to classification accuracy.

The stability results in Fig. 26 show that GPT-5.5 selects the same Top 1 technique across repeated runs in up to 59.1% of cases. The two GPT models exhibit comparable stability, Llama-4 performs slightly better with a *Same-Top-1* score of 68.2%, and both Qwen3 and Ministral are less consistent. The *Pair-Top-1* and *Fleiss-Kappa* measures confirm this ordering, indicating that GPT and Llama tend to repeatedly select the same technique, or a small set of recurring techniques, as

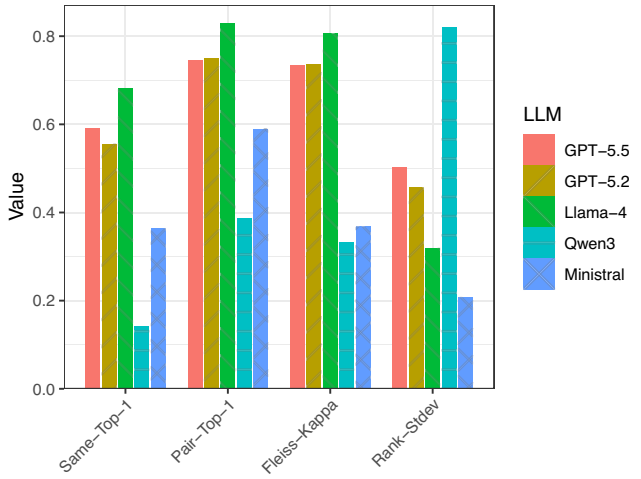


Fig. 26 Stability of LLM-based attack technique classification

Table 5 Goodman–Kruskal's Gamma of contingency tables

LLM	Pred. conf.	Comm. obs.	Event freq.	Alert level
GPT-5.5	0.45	0.48	0.01	0.31
GPT-5.2	0.42	0.44	0.10	0.35
Llama-4	0.21	0.34	0.07	0.34
Qwen3	0.23	0.23	0.05	0.21
Ministral	−0.37	−0.1	−0.12	−0.24

their top prediction. Finally, the *Rank-Stdev* values show that the position of the first correct prediction is relatively stable across runs, remaining at or below 0.5 for all models except Qwen3.

We analyze the relationship between classification accuracy and prediction confidence using GPT-5.5, the best-performing model in our evaluation. Figure 27 presents a contingency table covering the 198 techniques evaluated by the model. The vertical categories describe prediction performance: *Top 1* indicates techniques that were ranked first for all corresponding samples, *Top 2–10* indicates techniques outside of *Top 1* that appeared at least once among the Top 10 predictions, and *Outside Top 10* indicates techniques that never appeared among the Top 10 predictions. The horizontal categories represent the highest confidence level assigned to each technique across its samples. Most confidence assignments fall into the *Somewhat Certain: Attack* and *Almost Certain: Attack* categories, indicating that the model correctly classifies most attack samples above the *Neutral* level but only selects the highest confidence category in only a few cases. Six techniques were consistently ranked first and assigned the highest confidence level; these are network scanning (Scenario 1, Step 3, and Scenario 2, Step 21), exploitation for privilege escalation (Scenario 1, Step 36, and Scenario 7, Step 6), rootkit installation (Scenario 2, Step 27), and data encryption (Scenario 3, Step 26). Overall, the table indicates that higher confidence is associated with more accurate classifications. We quantify this relationship using Goodman–Kruskal's Gamma (γ) for ordinal categories. GPT-5.5 achieves $\gamma = 0.45$, with lower values observed for models with lower prediction accuracy, as shown in column *Pred. Conf.* of Table 5.

5.2.3 Impact of manifestation characteristics

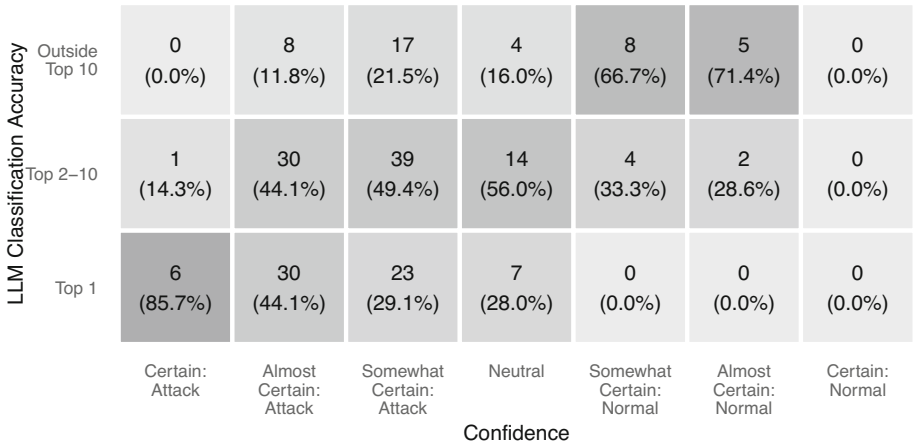
Finally, we investigate the relations between different types of attack manifestations and the LLM's classification performance. To this end, we correlate the three previously used buckets of prediction performance (*Top 1*, *Top 2–10*, and *Outside Top 10*) with previously analyzed manifestation characteristics, namely command observability (cf. Sect. 4.4.1), event frequency (cf. Sect. 4.4.2), and presence of IDS alerts (cf. Sect. 4.4.4).

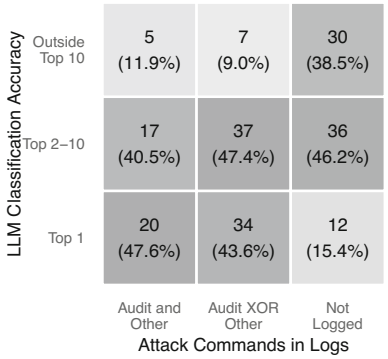
Figure 28 presents the results obtained from GPT-5.5 in the form of contingency tables and Table 5 contains γ for each combination of model and characteristic. Figure 28a shows that classification accuracy is highest when attack commands are visible in both audit logs and other log sources, slightly lower when commands appear in only one of these sources, and lowest when no command information is present in the logs analyzed by the LLM. Command observability has the strongest association with classification performance across all models, reaching $\gamma = 0.48$ for GPT-5.5.

Figure 28b indicates that attacks that generate either high (> 5000) or low (≤ 10) amounts of events are more accurately classified than those in between. Improved classification of attacks that involve many events may be attributable to the presence of stronger indicators within events of such attacks (cf. Fig. 15) and the informative nature of frequency patterns themselves. On the other hand, low-frequency attack steps have the advantage that a higher fraction of all events are sampled into the prompt, which could indicate a limitation of our sampling strategy. Because these effects do not follow a monotonic pattern, event frequency shows almost no ordinal association with classification performance, with $\gamma = 0.01$ for GPT-5.5 and similarly low values for the other models.

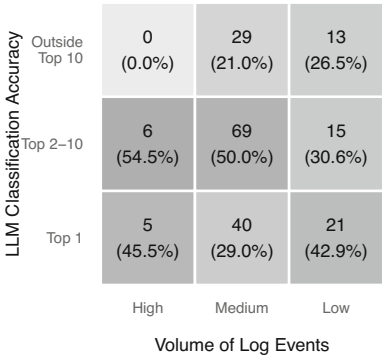
Finally, Fig. 28c shows that more attack steps triggering IDS alerts are within the Top 10 predictions compared to attacks that do not trigger alerts. This can be attributed either to the additional semantic information contained in the alerts themselves or to the fact that these attack steps are inherently more straightforward to identify as such and thus easier to interpret. The resulting association is moderate, with $\gamma = 0.31$ for GPT-5.5.

Fig. 27 LLM prediction confidence

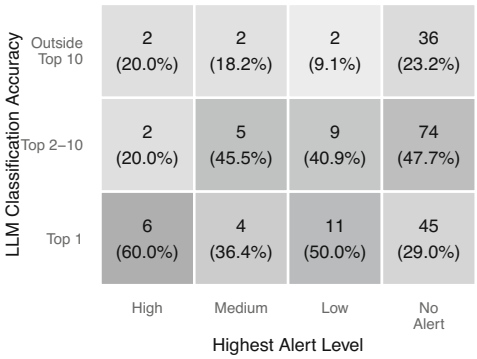




(a) Observability of commands.



(b) Log event frequencies.



(c) Highest alert severities.

Fig. 28 Contingency tables of LLM classification accuracies and attack manifestations

6 Discussion

In this section, we discuss the implications of our findings. We subsequently outline limitations of our work and derive recommendations for future research.

6.1 Implications

This paper introduces a new data set of cyber attack manifestations in system log data and provides an empirical analysis of their characteristics. Our results indicate that the majority of attack steps leave distinct traces in system logs that allow to reconstruct individual actions. Our analysis suggests that attack executions can be broadly grouped into at least four types of manifestations. First, attacks that leave no observable manifestations in common sources for system log data, which may occur for certain types of interactions as well as for preparatory steps that take place outside the monitored systems or network. Second, direct manifestations that expose executed commands. Across most scenarios, information about executed commands is present in the logs, despite the fact that attackers employ different execution mechanisms, including SSH (Scenarios 1, 3, and 4), implants (Scenarios 2 and 4), reverse-shells (Scenarios 1 and 3), VNC (Scenario 3), and screen sharing software (Scenario 6). While these mechanisms differ substantially in how attackers interact with compromised systems, our examples show that command information is often still observable. At the same time, the execution method strongly influences the granularity and visibility of these traces, with certain approaches effectively obscuring command-level details. Third, manifestations that do not reveal executed commands but are generated as direct consequences of malicious actions. Although less explicit, such events can still serve as strong indicators of adversary behavior, particularly when they stand out due to unusual parameters (e.g., events indicating system configuration changes) or high event frequencies (e.g., scanning activities). Fourth, indirect manifestations, such as changes in system performance metrics. These manifestations are generally more difficult to relate to specific attack techniques; nonetheless, unexpected changes of these metrics may be useful to detect undesired activities that are not captured or detectable in other log data.

Our results further show that while intrusion detection systems successfully trigger alerts for some attack steps, they fail to detect a large fraction of them. We acknowledge that

many attack commands closely resemble legitimate administrative actions, and that IDS rule sets intentionally avoid flagging such activities to limit false positives. Nonetheless, this observation highlights the limitations of purely signature-based detection and suggests the need for complementary analysis approaches.

One such complementary approach is LLM-based log interpretation. In our illustrative evaluation, the LLM classified 41.8% of attack steps with perfect accuracy and 67.2% within the Top 10 technique predictions. This result is particularly notable given the large search space of 216 MITRE ATT&CK techniques and the zero-shot setting, in which the model is provided with no contextual information beyond the log data generated during each attack step. These findings suggest that LLMs are in fact capable of extracting meaningful semantic patterns from system logs and have the potential to support automated log interpretation, thereby assisting human analysts in understanding attacker behavior.

6.2 Limitations and threats to validity

We acknowledge several limitations and threats to validity of our work. Our infrastructure for data collection is limited to Linux-based systems and open-source software, which we deliberately chose to ensure full reproducibility. However, attack manifestations may differ substantially on other operating systems such as Windows, where logging mechanisms, formats, and granularity, vary considerably. As a result, the performance of LLM-based log interpretation may differ significantly. Researchers should therefore complement our data set with Windows-based data sets such as the ones reviewed in Sect. 2.2 for evaluations.

Even though our methodology relies on idle systems and log filtering (cf. Sects. 3.1 and 4.1.1), we cannot guarantee that the collected data sets are entirely free of background noise. At the same time, our filtering procedure may remove events that are influenced by attacks. Some label inaccuracies may be introduced by delayed events that fall outside of their corresponding attack intervals or coincide with other attack steps. To mitigate these issues, we provide both filtered and unfiltered versions of the extracted attack manifestations (cf. Sect. 1). Moreover, despite independent manual validation of attack playbooks and the resulting attack manifestations, we cannot guarantee that every attack step completed as expected since not all steps produce explicit output.

We also emphasize that our evaluation of LLM-based log interpretation is intended as an illustrative case study rather than a comprehensive benchmark. Although we compare multiple LLMs, the observed results may not generalize to other models, particularly future model generations. Exact reproducibility of our results is also limited by model evolution, as externally hosted models may change over time even when the same prompts and parameters are used. Furthermore, limiting the prompt to ten randomly sampled events per log file means that the most informative events may not always be included, thereby affecting classification accuracy. Moreover, our evaluation strategy that counts correct classifications if any ground truth technique appears in the predicted list may bias results in favor of attack steps with many assigned techniques. For instance, system vulnerability scanning (Scenario 1, Step 13) is associated with 13 techniques, which increases the likelihood of obtaining at least one match in comparison to most attack steps which only involve one or two associated techniques. The absence of concurrent benign user activity during attack execution may also make the LLM-based interpretation task easier than in production environments, where attack-related events are interleaved with unrelated benign activity.

Finally, our attack scenarios are not derived from real-world incidents but are intentionally constructed to combine as many attack techniques as practically feasible into coherent kill chains. While scenario design was carried out by domain experts with experience in offensive technologies and attack modeling, these sequences may not fully reflect real attacker behavior. Specifically, the scenarios may not capture the full variability of human attacker behavior, such as redundant activities, extensive exploratory actions, unsuccessful attempts, adaptive changes in strategy, irregular timing, or individual attacker preferences. The minimum 15-second delay between attack steps (cf. Sect. 3.1.2) further introduces an artificial temporal separation that may affect methods relying on temporal event clustering and does not necessarily reflect the timing of real attacks. Moreover, although the scenarios and labels were independently cross-validated, their design and annotation by the same team may still have introduced unintentional bias.

6.3 Future work

Since the modeled attack cases are labeled at the technique level, the presented data set is directly applicable for evaluation of log interpretation approaches as demonstrated in our case study. Extending these labels could prove useful for evaluation of additional tasks for LLMs in the security domain, including log summarization [42, 48]. In particular, we assign natural-language descriptions to selected attack steps (cf. Fig. 4), which could serve as alternative or complementary ground truth labels. Future extensions could also incorporate labels related to mitigation strategies and recommended countermeasures, e.g., commands to remove backdoors and return the system into a secure state. We plan to release enriched versions of the data set with extended annotations as part of future publications.

In our illustrative evaluation, we only consider raw log data and IDS alerts as input to the LLM. However, additional contextual information could substantially improve

attack interpretation and classification. Specifically, our data set includes system configuration files collected from all hosts that contain details about installed software versions and system settings. Such asset information can provide valuable context that verifies or augments attack interpretations. For example, in the ZoneMinder exploit (Scenario 1, Step 5), the configuration files specify the exact version of the ZoneMinder application,[12] which is vulnerable to unauthenticated remote code execution according to public threat intelligence.[13] While the LLM is already capable of correctly interpreting the logs generated as consequences of this exploit (cf. Fig. 23) as command execution via the ZoneMinder web application (cf. Fig. 24a), the combination with asset data and external threat intelligence could further corroborate LLM interpretations, increase prediction confidence, enrich explanations with relevant vulnerability references, and potentially suggest actionable mitigation steps.

Another simplification of our evaluation is that we consider attack steps in isolation. In practice, correlating information across multiple steps would enable reasoning about the attacker's objectives and the overall progression of the kill chain. Instead of submitting independent prompts for each step, future work could incorporate log data or classifications from preceding attack steps as contextual information. This could resolve incorrect interpretations regarding benign or malicious intentions of underlying actions (cf. Fig. 24d). For example, commands executed by the attacker that are also commonly carried out by system administrators may appear as benign when considered in isolation, but are more likely to be interpreted correctly when immediately following other clearly malicious activities carried out by the same user or within the same context.

Furthermore, we use only a few general-purpose LLMs in this paper but recommend to compare performance across different models and architectures in future evaluations. In particular, prior work has shown that LLM fine-tuning can improve log interpretation capabilities [29]. Aside from that, given the already promising results obtained with a general-purpose model, it would be interesting to investigate whether smaller and cost-efficient models can achieve comparable performance [58]. Specifically, locally deployable models could address privacy concerns that potentially prohibit organizations from transferring log data to external LLM providers [23].

We also encourage future studies to compare LLM-based log interpretation with human expert analysis under controlled conditions using identical prompts and data. The resulting annotations could be used to validate the existing labels, improve our filtering mechanism, or to release updated versions of the data set with adapted or enriched labels. In particular, they could help identify attack steps for which reliable interpretation cannot be expected because the decisive attacker behavior is not sufficiently reflected in the observable log artifacts (cf. Sect. 4.4.1).

[12] We deploy ZoneMinder version 1.36.31 in our test environment.

[13] All ZoneMinder versions prior to 1.36.33 and 1.37.33 are vulnerable to the exploit considered in Scenario 1 [52].

7 Conclusion

In this paper, we demonstrated how CAM-LDS can support systematic and reproducible analysis of cyber attack manifestations across heterogeneous system logs and security alerts. By isolating the effects of individual attack steps in a controlled Linux-based environment, the data set enables detailed investigation of both direct and indirect evidence of adversarial behavior and provides a foundation for evaluating semantic log-analysis methods.

Our analysis revealed that attack manifestations vary substantially in their characteristics. While many attack steps leave explicit command-level traces in logs, others produce indirect indicators such as anomalous event frequencies, changes in system performance metrics, or intrusion detection alerts. At the same time, a considerable fraction of attack steps remain undetected by signature-based IDS, highlighting the limitations of rule-based detection approaches and the importance of complementary analysis methods such as LLMs.

To illustrate the utility of CAM-LDS, we conducted a zero-shot evaluation of LLM-based log interpretation. The results show that LLMs are capable of correctly identifying the underlying attack technique for a substantial portion of attack steps using raw log data alone. Classification performance appears connected to manifestation characteristics; in particular, observable attack commands and the presence of intrusion alerts are positively associated with classification accuracy for GPT-5.5 and several of the other evaluated models.

We release CAM-LDS together with the open-source infrastructure, attack scripts, and experimental artifacts to facilitate transparent, reproducible, and comparable research in the area of log-based security analysis. In particular, we encourage future work to incorporate additional contextual and asset information, as well as knowledge from previously classified attack steps, to further improve classification accuracy and more reliably distinguish malicious activity from benign behavior.

Author contributions All authors have contributed to this manuscript and approve of this submission. M.L. designed the overall concept and methodology, planned and carried out the evaluation, conducted the data analyses, created the visualizations, and wrote the original draft of the manuscript. W.H. and T.B. designed and implemented the attack scenarios, deployed the test environment and attack emulation engine, and collected the data. F.S. and M.W. acquired funding for this research activity and reviewed and edited the manuscript.

Funding Open access funding provided by AIT Austrian Institute of Technology GmbH. Funded by the European Union under the Horizon Europe Research and Innovation programme (GA no. 101168144 - MIRANDA). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them.

Data availability The data sets introduced in this paper are openly accessible: CAM-LDS system log data (https://zenodo.org/records/18390560) and CAM-LDS network packet captures (https://zenodo.org/records/18701094). We also provide artifacts of our case study (https://github.com/ait-aecid/attack-manifestations-interpretation) as well as code to deploy our virtual test environment (https://github.com/ait-testbed/attackbed).

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Alzu'bi, A., Darwish, O., Albashayreh, A., Tashtoush, Y.: Cyber-attack event logs classification using deep learning with semantic feature analysis. Comput. Secur. **150**, 104222 (2025). https://doi.org/10.1016/j.cose.2024.104222
2. Bagui, S.S., Mink, D., Bagui, S.C., Ghosh, T., Plenkers, R., McElroy, T., Dulaney, S., Shabanali, S.: Introducing uwf-zeekdata22: a comprehensive network traffic dataset based on the mitre att&ck framework. Data **8**(1), 18 (2023). https://doi.org/10.3390/data8010018
3. Balogh, Š, Mlynček, M., Vraňák, O., Zajac, P.: Using generative ai models to support cybersecurity analysts. Electronics **13**(23), 4718 (2024). https://doi.org/10.3390/electronics13234718
4. Beck, V., Landauer, M., Wurzenberger, M., Skopik, F., Rauber, A.: System log parsing with large language models: a review. (2025). arXiv:2504.04877 arXiv preprint
5. Berady, A., Jaume, M., Tong, V.V.T., Guette, G.: Pwnjutsu: a dataset and a semantics-driven approach to retrace attack campaigns. IEEE Trans. Netw. Serv. Manage. **19**(4), 5252–5264 (2022). https://doi.org/10.1109/TNSM.2022.3183476
6. Bönninghausen, P., Uetz, R., Henze, M.: Introducing a comprehensive, continuous, and collaborative survey of intrusion detection datasets. In: Proceedings of the 17th Cyber Security Experimentation and Test Workshop, pp. 34–40 (2024). https://doi.org/10.1145/3675741.3675754
7. CIRT.net: Nikto. https://cirt.net/nikto/ (2025). Accessed 29 June 2026
8. Collectd: The system statistics collection daemon. https://collectd.org/ (2026). Accessed 29 June 2026
9. Cotti, L., Drago, I., Rula, A., Bianchini, D., Cerutti, F.: Ontologx: ontology-guided knowledge graph extraction from cybersecurity logs with large language models. Adv. Intell. Syst. (2026). https://doi.org/10.1002/aisy.202501381
10. Cytopia: Smtp-user-enum. https://github.com/cytopia/smtp-user-enum (2022). Accessed 29 June 2026
11. Debian Project: Awffull. https://packages.debian.org/sid/web/awffull (2025). Accessed 29 June 2026
12. Duan, C., Jia, T., Yang, Y., Liu, G., Liu, J., Zhang, H., Zhou, Q., Li, Y., Huang, G.: Eagerlog: active learning enhanced retrieval augmented generation for log-based anomaly detection. In: ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 1–5 (2025). IEEE https://doi.org/10.1109/ICASSP49660.2025.10888663
13. Elam, M., Mink, D., Bagui, S.S., Plenkers, R., Bagui, S.C.: Introducing uwf-zeekdata24: an enterprise mitre att&ck labeled network attack traffic dataset for machine learning/ai. Data **10**(5), 59 (2025). https://doi.org/10.3390/data10050059
14. Ffuf contributors: Ffuf: Fuzz Faster U Fool. https://github.com/ffuf/ffuf (2026). Accessed 29 June 2026
15. Fleiss, J.L.: Measuring nominal scale agreement among many raters. Psychol. Bull. **76**(5), 378 (1971). https://doi.org/10.1037/h0031619
16. GitHub: Mordor. https://github.com/UraSecTeam/mordor (2019). Accessed 11 Feb 2026
17. GitHub: Evtx-attack-samples. https://github.com/sbousseaden/EVTX-ATTACK-SAMPLES (2020). Accessed 11 Feb 2026
18. GitHub: Evtx-to-mitre-attack. https://github.com/mdecrevoisier/EVTX-to-MITRE-Attack (2024). Accessed 11 Feb 2026
19. GitHub: atomic-evtx. https://github.com/arniki/atomic-evtx (2025). Accessed 11 Feb 2026
20. Goldschmidt, P., Chudá, D.: Network intrusion datasets: a survey, limitations, and recommendations. Comput. Secur. (2025). https://doi.org/10.1016/j.cose.2025.104510
21. Gordon Lyon: Nmap. https://nmap.org/ (2026). Accessed 29 June 2026
22. Gruntwork: Terragrunt. https://terragrunt.gruntwork.io/ (2026). Accessed 29 June 2026
23. Habibzadeh, A., Feyzi, F., Atani, R.E.: Large language models for security operations centers: a comprehensive survey. arXiv preprint arXiv:2509.10858 (2025). https://doi.org/10.1155/jece/3383674
24. Hans, S., Marsella, S., Hirschmann, S., Gurney, N.: Security logs to att&ck insights: leveraging llms for high-level threat understanding and cognitive trait inference. arXiv preprint arXiv:2510.20930 (2025)
25. Hauser, V.: Hydra. https://github.com/vanhauser-thc/thc-hydra (2023). Accessed 29 June 2026
26. He, S., Zhu, J., He, P., Lyu, M.R.: Experience report: System log analysis for anomaly detection. In: 2016 IEEE 27th international symposium on software reliability engineering (ISSRE), pp. 207–218. IEEE (2016). https://doi.org/10.1109/ISSRE.2016.21
27. Hotwagner, W.: Logrotten. https://github.com/whotwagner/logrotten (2022). Accessed 29 June 2026
28. Information technology - Security techniques - Network security - Part 3: Reference networking scenarios - Threats, design techniques and control issues. Standard, International Organization for Standardization, (2010)
29. Ji, Y., Liu, Y., Yao, F., He, M., Tao, S., Zhao, X., Su, C., Yang, X., Meng, W., Xie, Y., et al.: Adapting large language models to log analysis with interpretable domain knowledge. In: Proceedings of the 34th ACM International Conference on Information and Knowledge Management, pp. 1135–1144 (2025). https://doi.org/10.1145/3746252.3761189

30. Ju, H.: Reliable online log parsing using large language models with retrieval-augmented generation. In: 2024 IEEE 35th International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 99–102. IEEE (2024). https://doi.org/10.1109/ISSREW63542.2024.00055
31. Karim, S.S., Afzal, M., Iqbal, W., Al Abri, D.: Advanced persistent threat (apt) and intrusion detection evaluation dataset for linux systems 2024. Data Brief **54**, 110290 (2024). https://doi.org/10.1016/j.dib.2024.110290
32. Karlsen, E., Luo, X., Zincir-Heywood, N., Heywood, M.: Benchmarking large language models for log analysis, security, and interpretation. J. Netw. Syst. Manage. **32**(3), 59 (2024). https://doi.org/10.1007/s10922-024-09831-x
33. Khraisat, A., Gondal, I., Vamplew, P., Kamruzzaman, J.: Survey of intrusion detection systems: techniques, datasets and challenges. Cybersecurity **2**(1), 1–22 (2019). https://doi.org/10.1186/s42400-019-0038-7
34. Kilian, S., Tong, V.V.T., Lalande, J.F., Majorczyk, F., Sanchez, A., Talon, N., Besson, P.V., Orsini, H., Lledo, P., Gimenez, P.F.: Casinolimit: An offensive dataset labeled with mitre att&ck techniques. In: The 28th International Symposium on Research in Attacks, Intrusions and Defenses (RAID) (2025). https://doi.org/10.1109/RAID67961.2025.00039
35. Landauer, M., Hotwagner, W., Boenke, T., Skopik, F., Wurzenberger, M.: Attackmate: realistic emulation and automation of cyber attack scenarios across the kill chain, (2026). arXiv:2601.14108 arXiv preprint
36. Landauer, M., Skopik, F., Frank, M., Hotwagner, W., Wurzenberger, M., Rauber, A.: Maintainable log datasets for evaluation of intrusion detection systems. IEEE Trans. Dependable Secure Comput. **20**(4), 3466–3482 (2022). https://doi.org/10.1109/TDSC.2022.3201582
37. Landauer, M., Skopik, F., Wurzenberger, M.: Introducing a new alert data set for multi-step attack analysis. In: Proceedings of the 17th Cyber Security Experimentation and Test Workshop, pp. 41–53 (2024). https://doi.org/10.1145/3675741.3675748
38. Landauer, M., Skopik, F., Wurzenberger, M., Hotwagner, W., Rauber, A.: Have it your way: generating customized log datasets with a model-driven simulation testbed. IEEE Trans. Reliab. **70**(1), 402–415 (2020). https://doi.org/10.1109/TR.2020.3031317
39. Landauer, M., Wurzenberger, M., Skopik, F., Hotwagner, W., Höld, G.: Aminer: a modular log data analysis pipeline for anomaly-based intrusion detection. Digital Threats: Res. Practice **4**(1), 1–16 (2023). https://doi.org/10.1145/3567675
40. Laprade, C., Bowman, B., Huang, H.H.: Picodomain: a compact high-fidelity cybersecurity dataset, (2020). arXiv:2008.09192 arXiv preprint
41. Linux: Logrotate. https://linux.die.net/man/8/logrotate (2026). Accessed 29 June 2026
42. Liu, Y., Ji, Y., Tao, S., He, M., Meng, W., Zhang, S., Sun, Y., Xie, Y., Chen, B., Yang, H.: Loglm: from task-based to instruction-based automated log analysis. In: 2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pp. 401–412. IEEE (2025). https://doi.org/10.1109/ICSE-SEIP66354.2025.00041
43. Liu, Y., Tao, S., Meng, W., Wang, J., Ma, W., Chen, Y., Zhao, Y., Yang, H., Jiang, Y.: Interpretable online log analysis using large language models with prompt strategies. In: Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension, pp. 35–46 (2024). https://doi.org/10.1145/3643916.3644408
44. Liu, Y., Tao, S., Meng, W., Yao, F., Zhao, X., Yang, H.: Logprompt: prompt engineering towards zero-shot and interpretable log analysis. In: Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings, pp. 364–365 (2024). https://doi.org/10.1145/3639478.3643108
45. Mabrouk, A., Hatem, M., Mamun, M., Saad, S.: Lmdg: advancing lateral movement detection through high-fidelity dataset generation. (2025). arXiv:2508.02942 arXiv preprint
46. Medenou, R.D., Mayo, V.M.C., Balufo, M.G., Castrillo, M.P., Garrido, F.J.G., Martinez, A.L., Catalán, D.N., Hu, A., Rodríguez-Bermejo, D.S., Vidal, J.M., et al.: Cysas-s3: a novel dataset for validating cyber situational awareness related tools for supporting military operations. In: Proceedings of the 15th International Conference on Availability, Reliability and Security, pp. 1–9 (2020). https://doi.org/10.1145/3407023.3409222
47. Mohamed, M.S., Arabo, A.: A siem-integrated cybersecurity prototype for insider threat anomaly detection using enterprise logs and behavioural biometrics. Electronics **15**(1), 248 (2026). https://doi.org/10.3390/electronics15010248
48. Mudgal, P., Wouhaybi, R.: An assessment of chatgpt on log data. In: International Conference on AI-generated Content, pp. 148–169. Springer (2023). https://doi.org/10.1007/978-981-99-7587-7_13
49. Myneni, S., Chowdhary, A., Sabur, A., Sengupta, S., Agrawal, G., Huang, D., Kang, M.: Dapt 2020-constructing a benchmark dataset for advanced persistent threats. In: International workshop on deployable machine learning for security defense, pp. 138–163. Springer (2020). https://doi.org/10.1007/978-3-030-59621-7_8
50. Myneni, S., Jha, K., Sabur, A., Agrawal, G., Deng, Y., Chowdhary, A., Huang, D.: Unraveled: a semi-synthetic dataset for advanced persistent threats. Comput. Netw. **227**, 109688 (2023). https://doi.org/10.1016/j.comnet.2023.109688
51. National Institute of Standards and Technology: CVE-2021-4034. https://nvd.nist.gov/vuln/detail/CVE-2021-4034 (2022). Accessed 29 June 2026
52. National Institute of Standards and Technology: CVE-2023-26035. https://nvd.nist.gov/vuln/detail/CVE-2023-26035 (2023). Accessed 29 June 2026
53. Ndichu, S., Yamada, A., Ban, T., Ozawa, S., Takahashi, T., Inoue, D.: Ai-driven security alert screening and alert fatigue mitigation in security operations centers: a comprehensive survey (2026). arXiv:2605.08316 arXiv preprint
54. Nextcloud contributors: Nextcloud. https://github.com/nextcloud (2026). Accessed 29 June 2026
55. Oliner, A., Stearley, J.: What supercomputers say: a study of five system logs. In: 37th annual IEEE/IFIP international conference on dependable systems and networks (DSN'07), pp. 575–584. IEEE (2007). https://doi.org/10.1109/DSN.2007.103
56. Open Information Security Foundation: Suricata. https://suricata.io/ (2026). Accessed 29 June 2026
57. OpenTofu: Open-Source Infrastructure as Code. https://opentofu.org/ (2026). Accessed 29 June 2026
58. Palma, G., Cecchi, G., Caronna, M., Rizzo, A.: Leveraging large language models for scalable and explainable cybersecurity log analysis. J. Cybersecur. Privacy **5**(3), 55 (2025). https://doi.org/10.3390/jcp5030055
59. Pan, J., Liang, W.S., Yidi, Y.: Raglog: Log anomaly detection using retrieval augmented generation. In: 2024 IEEE World Forum on Public Safety Technology (WFPST), pp. 169–174. IEEE (2024). https://doi.org/10.1109/WFPST58552.2024.00034
60. PEASS-ng contributors: LinPEAS: Linux Privilege Escalation Awesome Script. https://github.com/peass-ng/PEASS-ng/tree/master/linPEAS (2026). Accessed 29 June 2026
61. Pérez-Sánchez, A., Palacios, R., López, G.L.: Comiset: dataset for the analysis of malicious events in windows systems. Data Brief (2025). https://doi.org/10.1016/j.dib.2025.111723
62. Puppet contributors: Puppet. https://github.com/puppetlabs/puppet (2026). Accessed 29 June 2026
63. Purslane Tech Pte. Ltd.: RustDesk. https://rustdesk.com/ (2026). Accessed 29 June 2026
64. PwnKit contributors: PwnKit. https://github.com/ly4k/PwnKit (2022). Accessed 29 June 2026

65. Qi, J., Huang, S., Luan, Z., Yang, S., Fung, C., Yang, H., Qian, D., Shang, J., Xiao, Z., Wu, Z.: Loggpt: exploring chatgpt for log-based anomaly detection. In: 2023 IEEE International Conference on High Performance Computing & Communications, Data Science & Systems, Smart City & Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys), pp. 273–280. IEEE (2023). https://doi.org/10.1109/HPCC-DSS-SmartCity-DependSys60770.2023.00045
66. Rahal, K., Riahi, A., Debatty, T.: Dataset of apt persistence techniques on windows platforms mapped to the mitre att&ck framework. In: 2025 28th Conference on Innovation in Clouds, Internet and Networks (ICIN), pp. 17–24. IEEE (2025). https://doi.org/10.1109/ICIN64016.2025.10943025
67. Rahman, M.R., Basak, S.K., Hezaveh, R.M., Williams, L.: Attackers reveal their arsenal: an investigation of adversarial techniques in cti reports. (2024). arXiv:2401.01865 arXiv preprint
68. Rapid7: Nextcloud Workflows Remote Code Execution. https://www.rapid7.com/db/modules/exploit/unix/webapp/nextcloud_workflows_rce/ (2025). Accessed 29 June 2026
69. Rapid7: Metasploit. https://www.metasploit.com/ (2026). Accessed 29 June 2026
70. Red Hat: Ansible. https://www.ansible.com/ (2026). Accessed 29 June 2026
71. Riddle, A., Westfall, K., Bates, A.: Atlasv2: atlas attack engagements, version 2, (2023). arXiv:2401.01341 arXiv preprint
72. Shanto, S.S., Paul, R., Ahmed, Z., Reza, A.S., Islam, K.M., Roy, S.S.: Console log explainer: a framework for generating automated explanations using llm. In: 2024 2nd International Conference on Artificial Intelligence, Blockchain, and Internet of Things (AIBThings), pp. 1–5. IEEE (2024). https://doi.org/10.1109/AIBThings63359.2024.10863559
73. Sharafaldin, I., Lashkari, A.H., Ghorbani, A.A.: Toward generating a new intrusion detection dataset and intrusion traffic characterization. ICISSP **1**, 108–116 (2018). https://doi.org/10.5220/0006639801080116
74. Shen, X., Li, Z., Burleigh, G., Wang, L., Chen, Y.: Decoding the mitre engenuity att&ck enterprise evaluation: an analysis of edr performance in real-world environments. In: Proceedings of the 19th ACM Asia conference on computer and communications security, pp. 96–111 (2024). https://doi.org/10.1145/3634737.3645012
75. Singh, R., Tariq, S., Jalalvand, F., Chhetri, M.B., Nepal, S., Paris, C., Lochner, M.: Llms in the soc: An empirical study of human-ai collaboration in security operations centres. (2025). arXiv:2508.18947 arXiv preprint
76. Srinivas, S., Kirk, B., Zendejas, J., Espino, M., Boskovich, M., Bari, A., Dajani, K., Alzahrani, N.: Ai-augmented soc: a survey of llms and agents for security automation. J. Cybersecur. Privacy **5**(4), 95 (2025). https://doi.org/10.3390/jcp5040095
77. Stühn, J., Hilgert, J.N., Lambertz, M.: The hidden threat: analysis of linux rootkit techniques and limitations of current detection tools. Digital Threats Res. Practice **5**(3), 1–24 (2024). https://doi.org/10.1145/3688808
78. Sun, D., Zhang, J., Xu, J., Zheng, Y., Tian, Y., Li, Z.: From alerts to intelligence: a novel llm-aided framework for host-based intrusion detection. (2025). arXiv:2507.10873 arXiv preprint
79. Tejero-Fernández, J.J., Sánchez-Macián, A.: Evaluating language models for threat detection in iot security logs. (2025). arXiv:2507.02390 arXiv preprint
80. The Document Foundation: LibreOffice. https://www.libreoffice.org/ (2026). Accessed 29 June 2026
81. The MITRE Corporation: Common Attack Pattern Enumeration and Classification (CAPEC). https://capec.mitre.org/ (2026). Accessed 29 June 2026
82. The Zeek Project: The Zeek Network Security Monitor. https://zeek.org/ (2026). Accessed 29 June 2026
83. Turcotte, M., Labrèche, F., Paquette, S.O.: Automated alert classification and triage (aact): an intelligent system for the prioritisation of cybersecurity alerts. (2025). arXiv:2505.09843 arXiv preprint
84. VeilTransfer contributors: VeilTransfer. https://github.com/infosecn1nja/VeilTransfer (2024). Accessed 29 June 2026
85. Waeytens, F.: Dnsenum. https://github.com/fwaeytens/dnsenum (2015). Accessed 29 June 2026
86. Wazuh: The Open Source Security Platform. https://wazuh.com/ (2026). Accessed 29 June 2026
87. Wolf, M., Bergner, K., Förtsch, P., Landes, D.: Implications of applying emulation data to machine learning-based intrusion detection systems. J. Inf. Secur. Appl. **97**, 104362 (2026). https://doi.org/10.1016/j.jisa.2025.104362
88. Zha, J., Shan, X., Lu, J., Zhu, J., Liu, Z.: Leveraging large language models for efficient alert aggregation in aiops. Electronics **13**(22), 4425 (2024). https://doi.org/10.3390/electronics13224425
89. Zhang, L., Jia, T., Jia, M., Wu, Y., Liu, H., Li, Y.: Xraglog: a resource-efficient and context-aware log-based anomaly detection method using retrieval-augmented generation. In: AAAI 2025 Workshop on Preventing and Detecting LLM Misinformation (PDLM) (2025)
90. Zhang, W., Zhang, Q., Yu, E., Ren, Y., Meng, Y., Qiu, M., Wang, J.: Leveraging rag-enhanced large language model for semi-supervised log anomaly detection. In: 2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE), pp. 168–179. IEEE (2024). https://doi.org/10.1109/ISSRE62328.2024.00026
91. Zhang, Z., Li, S., Zhang, L., Ye, J., Hu, C., Yan, L.: Llm-lade: large language model-based log anomaly detection with explanation. Knowl.-Based Syst. **326**, 114064 (2025). https://doi.org/10.1016/j.knosys.2025.114064
92. Zhu, J., He, S., He, P., Liu, J., Lyu, M.R.: Loghub: a large collection of system log datasets for ai-driven log analytics. In: 2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE), pp. 355–366. IEEE (2023). https://doi.org/10.1109/ISSRE59848.2023.00071
93. ZoneMinder: Open Source Video Surveillance. https://zoneminder.com/ (2026). Accessed 29 June 2026

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.