LMDG: Advancing Lateral Movement Detection Through High-Fidelity Dataset Generation

Anas Mabrouk, Mohamed Hatem, Mohammad Mamun, *Senior Member, IEEE*, Sherif Saad, *Senior Member, IEEE*

***Abstract*—Lateral Movement (LM) attacks continue to pose a significant threat to enterprise security, enabling adversaries to stealthily compromise critical assets. However, the development and evaluation of LM detection systems are impeded by the absence of realistic, well-labeled datasets. To address this gap, we propose *LMDG*, a reproducible and extensible framework for generating high-fidelity LM datasets. *LMDG* automates benign activity generation, multi-stage attack execution, and comprehensive labeling of system and network logs, dramatically reducing manual effort and enabling scalable dataset creation. A central contribution of *LMDG* is *Process Tree Labeling*, a novel agent-based technique that traces all malicious activity back to its origin with high precision. Unlike prior methods such as *Injection Timing* or *Behavioral Profiling*, *Process Tree Labeling* enables accurate, step-wise labeling of malicious log entries, correlating each with specific attack step and *MITRE ATT&CK* TTPs. To our knowledge, this is the first approach to support fine-grained labeling of multi-step attacks, providing critical context for detection models such as attack path reconstruction. We used *LMDG* to generate a 25-day dataset within a 25-VM enterprise environment containing 22 user accounts. The dataset includes 944 GB of host and network logs and embeds 35 multi-stage LM attacks, with malicious events comprising less than 1% of total activity—reflecting realistic benign-to-malicious ratio for evaluating detection systems. *LMDG* generated datasets improves upon existing ones by offering diverse LM attacks, up-to-date attack patterns, longer attacks timeframes, comprehensive data sources, realistic network architectures, and more accurate labeling.**

***Index Terms*—Advanced Persistent Threats (APTs), Lateral Movement (LM), cybersecurity benchmarks, multi-stage attacks, *MITRE Att&ck*.**

I. Introduction

ADVANCED Persistent Threats (APTs) represent a sophisticated category of cyberattacks characterized by their prolonged and stealthy presence within a targeted computer system or network, aimed at ultimately exfiltrating sensitive data or causing significant harm [2], [18], [48]. APTs employ a diverse array of techniques and tactics meticulously crafted to circumvent the defensive mechanisms of the victim's security infrastructure [1]. Among the array of sophisticated techniques employed by advanced threat actors, the concept of LM has emerged as a critical strategy for adversaries seeking to maneuver within compromised network environments. As elucidated by the exposition in [1], LM embodies an array of methodologies engaged by malevolent entities to infiltrate and orchestrate control over remote network systems. The attainment of their intended goals is frequently characterized by the imperative act of pivoting across an assortment of interconnected systems and accounts. Corresponding definitions mirroring this conception of LM are also extant within the literature, as expounded upon in [24], [4], [33], and [42], delineating the concept as the orchestrated movement of an attacker from a primary host to successive nodes within a compromised network, culminating in the pursuit of a valuable target.

LM-based attacks are becoming a growing threat to large private and government networks, frequently causing information exfiltration and service disruptions [10]. Analyzing various APT campaigns reveals that nearly all employ LM to navigate networks. The purpose of LM is to transition from one system to another, infiltrating additional resources and gaining higher privileges. This process enables attackers to discover and collect valuable data, expand their control over the targeted organization, and maintain long-term access to the compromised IT infrastructure [2], [18], [45], [51]. Since LM is a crucial phase in an APT attack, early detection is vital to minimize losses and prevent attackers from gaining further access to the network [9].

Detecting LM attacks poses a significant challenge, primarily due to several factors; firstly, the prolonged duration of these attacks, which can extend over months, significantly complicates their detection. Additionally, the sheer volume of enterprise traffic provides adversaries ample opportunities to blend in and seamlessly remain undetected amidst regular network activity. Various tactics and techniques exist for executing LM attacks, often leaving traces within network and system logs [1]. Attackers can effectively evade detection mechanisms by leveraging legitimate authentication credentials, system tools, and other evasion techniques. Furthermore, the prevalence of false security alerts further adds to the difficulty of distinguishing genuine threats from benign anomalies. Moreover, the incorporation of zero-day exploits or novel malware variants as part of these attacks further amplifies the complexity of detection [5], [7], [9], [11], [23].

Current research endeavors for LM detection rely mainly on machine learning [10], [32], [34], [47], [49]. The machine learning paradigm depends heavily on datasets to train and evaluate detection models, and the quality of these datasets directly impacts model performance and evaluation accuracy. Without high-quality training data, models can exhibit performance discrepancies, reducing accuracy and increasing false positives [16], [26]. A growing body of literature explores the evidence supporting that neglecting the fundamental importance of data has led to inaccuracies and bias in ML models [37]. For instance, researchers in [13] demonstrated that

This paper is based on the author's MSc thesis; for more details, see [35].

even minor modifications to a benchmark dataset significantly impact model performance more than the specific machine learning technique. Therefore, better data quality is essential to improve generalization and avoid bias in machine learning models [6], [44].

Most cybersecurity datasets suffer from quality issues, particularly those containing LM attacks. Common data quality problems include noisy labels, insufficient labeling, class imbalance, limited diversity of attack patterns, outdated attack types, simplistic synthetic generation environments, and short generation periods (see section V). Additionally, many existing datasets either lack instances of LM attacks altogether or contain only a limited number of such instances [15], [21], [26], [50]. Consequently, developing a comprehensive dataset, or ideally a framework, that addresses these challenges and others is essential for advancing research in LM detection.

To this end, our paper introduces a framework called LMDG (LM Datasets Generator), which addresses most of the issues discussed in section V. Our contributions can be summarized as follows:

- We analyzed existing cybersecurity datasets for LM attack properties, including techniques, time frames, movement hops, data sources, labeling methods, and testbed designs. We also reviewed frameworks for creating LM datasets V, making this the first study focused on LM dataset evaluation.

- Creating a benchmark dataset focused on LM attacks that address many of the existing issues in current LM datasets and conducting a qualitative analysis of it III IV. This dataset will be valuable for the research community in training and evaluating LM detection models.

- We developed the LMDG framework to generate reproducible LM/APT datasets (see Section II). It automates benign data generation II-C, attack execution II-D, and labeling. Automatic labeling is challenging in LM datasets due to malicious activities by benign hosts. Existing techniques include Injection Timing, Behavior Profiles, and Network Security Tools [21], [28]. We introduce a new method, ***process tree labeling***, which we argue is the most accurate II-E.

II. LMDG Framework

A. Overview

The LMDG framework uses VirtualBox to simulate organizational networks for LM detection research. While synthetic datasets offer controlled conditions and flexible attack simulations, they may lack realism. To address this, LMDG employs advanced virtualization techniques to enhance dataset fidelity. VirtualBox's networking options—NAT, Bridged, Internal, and Host-Only networks—allow for flexible and realistic network simulations. These configurations help emulate diverse organizational topologies, improving the practicality of the generated datasets for cybersecurity research.

Active Directory (AD) is a widely used directory service for managing IT profiles, enabling authentication, authorization, and resource management [38], [40]. The recent CrowdStrike incident highlighted the critical reliance on Windows systems, causing major disruptions and financial losses [27], [41]. The LMDG framework integrates Windows Domains and AD to create realistic network environments for cybersecurity research, improving the accuracy of simulated datasets for studying advanced threats like APTs.

The LMDG framework uses Wireshark for capturing detailed network traffic traces and employs a persistent logging service on all hosts and gateways to ensure data continuity. It also leverages Windows Event Logs for comprehensive system and security event records. This combination ensures high-fidelity data collection, crucial for developing effective cybersecurity detection models.

B. Testbed Infrastructure

The network, shown in Figure 1, simulates a small company with five departments, each in its own subnet with a dedicated Windows domain. For example, the Sales department resides in *sales.lmt.com* on subnet *192.168.59.0/24* with domain controller *DC 3*. Additional subnets include the root domain *lmt.com*, company servers, and a DMZ (*192.168.0.0/24*) within the IT domain. Routers connect these segments, and the structure can be adjusted as needed.

The setup uses VirtualBox with internal networks for subnets (isolated from external traffic) and a NAT network for the DMZ to enable controlled external communication. Hosts run Windows 10/11, while servers use Windows Server 2022 to reflect modern enterprise environments.

This topology improves realism by mirroring real-world enterprise networks with multiple subnets, dedicated domains, and a DMZ for public-facing services. These features enhance dataset fidelity, making it valuable for cybersecurity research.

C. Benign Data Engine (BDE)

In the context of the LMDG framework, the Benign Data Engine (BDE) is tasked with generating normal network behavior, effectively simulating employee activities. Figure 2 provides an overview of the BDE engine, which comprises two primary components: the **Sessions Scheduler** II-C1 and the **Sessions Executor** II-C2. The engine operates based on four key inputs:

- **User Credentials and Hosts:** This includes the credentials of employees and the specific hosts (workstations or devices) they use within the network.
- **Sessions Scheduler Configuration File:** This file defines the parameters for the Sessions Scheduler, dictating how it should generate and manage sessions timing for each user or employee.
- **Behavioral Scripts:** These scripts detail the activities of employees, it can operate on both individual level and a departmental level, such as those specific to the IT department. They encapsulate routine tasks and behaviors expected in a typical workday.

The Sessions Scheduler orchestrates generating session behaviors (i.e., login and logout times), ensuring the simulated

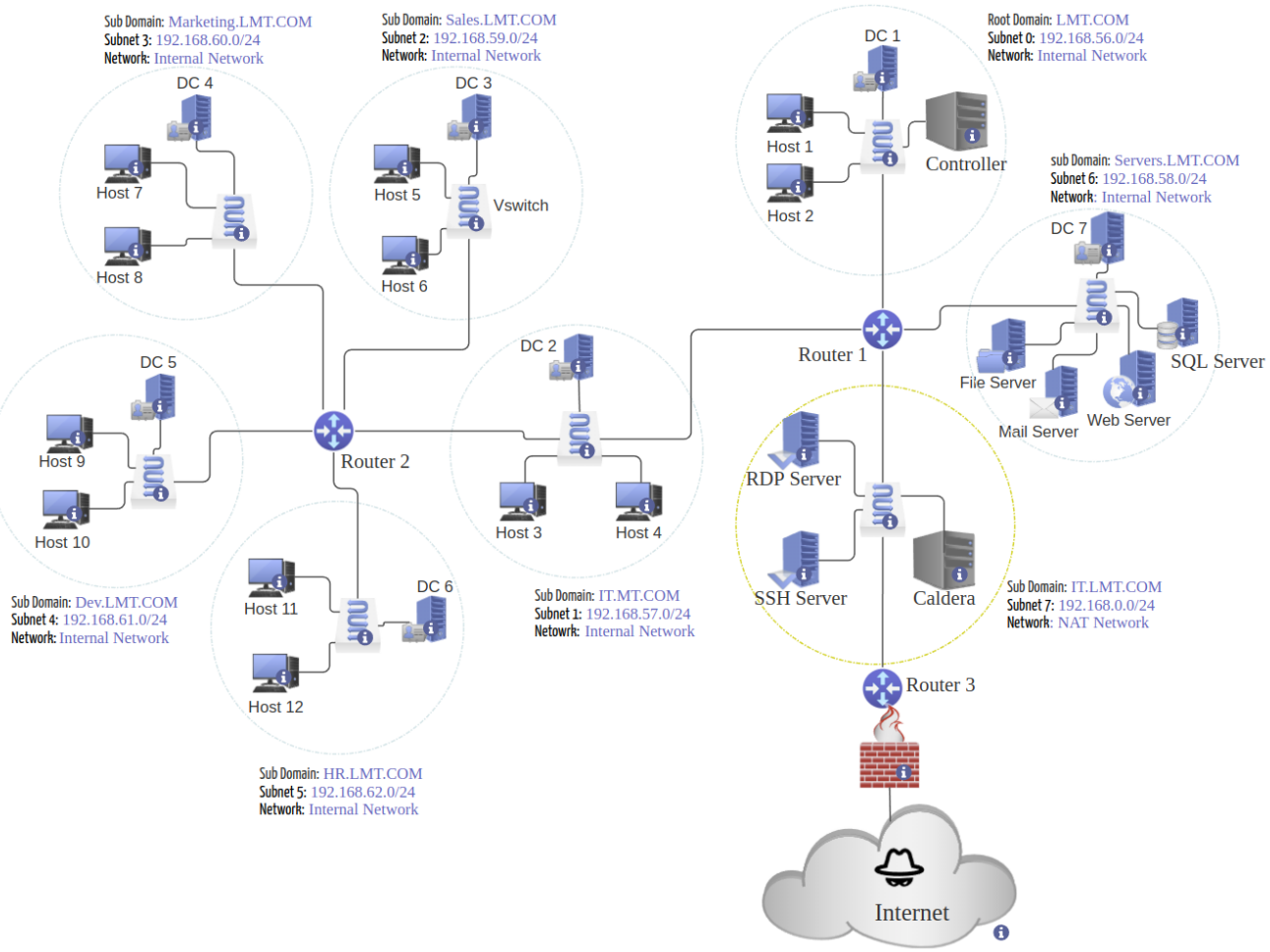

Fig. 1: The Testbed architecture used to generate LMDG dataset.

activities align with realistic standard user behavior patterns. Concurrently, the Sessions Executor enables the efficient simulation of multiple user sessions, reflecting the concurrent activities of various employees within the network. This design enhances the generated data's realism and ensures scalability and performance in simulating complex network environments.

*1) **Sessions Scheduler**:* The role of the Sessions Scheduler is to generate a list of tuples $\mathcal{S}_i = [(t_1, t_2), (t_3, t_4), \ldots, (t_{2k_i-1}, t_{2k_i})]$ for each employee i, representing their session behavior. Each tuple (t_{2j-1}, t_{2j}) denotes the login and logout times of a session, where t_{2j-1} is the login time and t_{2j} is the logout time. The duration of a session is given by $t_{2j} - t_{2j-1}$. The final output of the Sessions Scheduler is a list of lists $\mathcal{T} = [\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n]$, where $\mathcal{T}$ encapsulates the session behaviors for all n employees in the network. Each $\mathcal{S}_i$ in $\mathcal{T}$ provides a detailed account of an individual employee's login and logout activities throughout the day.

The process by which the Sessions Scheduler generates the lists $\mathcal{S}_i$ for an employee i is outlined as follows. Initially, the Sessions Scheduler determines whether employee i is absent based on probability values specified in the configuration file (third input, Figure 2) defined by the dataset creators. For instance, the dataset creators can define a probability interval $[p_1, p_2]$, where $0 \leq p_1, p_2 \leq 1$. The Sessions Scheduler then selects a random value from this interval to represent the probability of employee i being absent on a given day. This approach ensures that each employee i has a distinct probability of being absent. If employee i is absent, then the list $\mathcal{S}_i = \varnothing$. Additionally, there is a separate probability interval $[p_1', p_2']$ for determining absences during weekends, which typically corresponds to a higher probability.

If employee i is not absent, the Sessions Scheduler will proceed to generate the list $\mathcal{S}_i$. Initially, it determines the starting time (first login) for employee i. To facilitate this, the dataset creators define four *time intervals* representing various starting times: abnormally early, abnormally late, late, and on time. These intervals are denoted as $[t_{e1}, t_{e2}]$, $[t_{a1}, t_{a2}]$, $[t_{l1}, t_{l2}]$, and $[t_{o1}, t_{o2}]$ respectively. To determine the four possible starting times for the current employee i, the Sessions Scheduler randomly selects a value from each of the four corresponding intervals. Thus, for employee i, there exist four distinct candidate starting times denoted as $t_{\text{start\_abnormal\_early}}$, $t_{\text{start\_abnormal\_late}}$, $t_{\text{start\_late}}$, and $t_{\text{start\_on\_time}}$. In the configuration file, operators can define different probability intervals for each possible starting time, namely $[p_{e1}, p_{e2}]$, $[p_{a1}, p_{a2}]$, $[p_{l1}, p_{l2}]$,

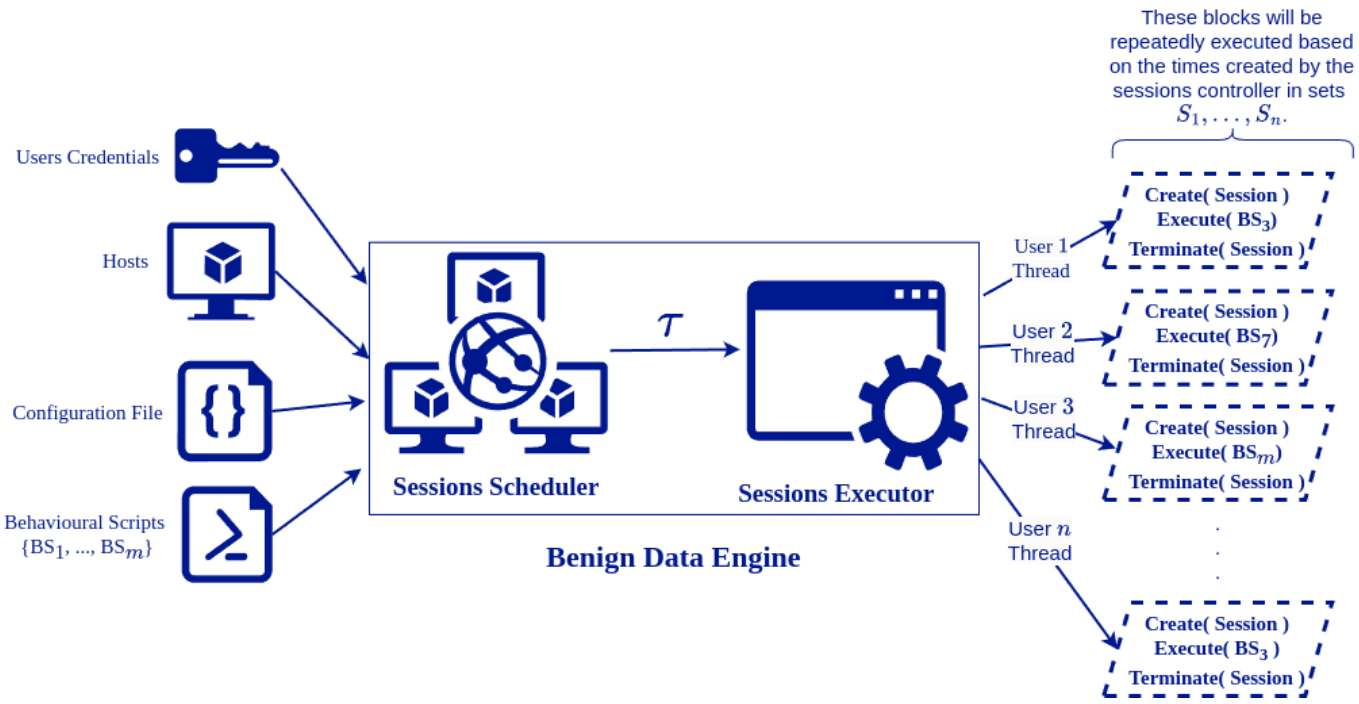

Fig. 2: Benign Data Engine (BDE) overview.

and $[p_{o1}, p_{o2}]$. It is noteworthy that drawing a probability value from intervals $[p_{e1}, p_{e2}]$ and $[p_{a1}, p_{a2}]$ will be typically very small, reflecting the rarity of abnormally early and abnormally late starting times. Conversely, the interval $[p_{o1}, p_{o2}]$ will produce the highest probabilities, indicating the likelihood of employees starting on time. Consequently, the Sessions Scheduler assigns a random probability value to each candidate starting time, drawn from their respective probability intervals. The next step can be linked to tossing an unfair tetrahedron (a die with four faces), where each face represents a starting time option. The resulting face corresponds to the actual starting time of employee i, denoted as t_{start}, which constitutes the first value of the first tuple in the list $\mathcal{S}_i$, i.e., t_1. Thus, the Sessions Scheduler effectively determines the starting time for employee i using this probabilistic method, ensuring that each potential starting time is considered.

Selecting the end time t_{end} for each employee i, denoted as the second time in the last tuple of the list $\mathcal{S}_i$ (i.e., last logout time t_{2k_i}), undergoes a process akin to determining the start time t_{start}. Similarly, the Sessions Scheduler employs a probabilistic approach, mirroring the methodology used for selecting t_{start}. Dataset creators define intervals representing various end times, such as abnormally early, abnormally late, late, and on time, each associated with corresponding probability intervals.

The Sessions Scheduler is not limited to drawing values from the defined time and probability intervals using a uniform distribution; it can also utilize exponential and normal distributions. For instance, consider Figure 3, which illustrates the Sessions Scheduler's process of selecting the value for $t_{\text{start\_abnormal\_early}}$ over 20,000 iterations. In this example, the Sessions Scheduler is configured to draw a time value t within the interval [3:30 AM - 7:29 AM] according to an exponential distribution with a lambda $\lambda = 0.00037$, where λ is the distribution parameter.

After defining t_{start} and t_{end}, the Sessions Scheduler will determine whether employee i will have a lunch break using a similar probabilistic approach. If a lunch break is scheduled, the controller will then specify t_{lunch_start} and t_{lunch_end}, which denote the start and end times of the lunch break, respectively.

The Sessions Scheduler manages random logouts and logins between t_{start} and t_{lunch_start}, and between t_{lunch_end} and t_{end}. This process occurs in two stages: first, the number and duration of logouts are randomly selected based on predefined intervals set in the configuration file (see Figure 2). Next, an algorithm recursively places these logouts on the timeline while adhering to minimum and maximum gap constraints between consecutive logouts also defined in the configuration file.

*2) **Sessions Executor**:* The Sessions Executor, a component of the Benign Data Engine (Figure 2), manages session execution by creating a thread for each employee i in the list $\mathcal{T}$. Using employee credentials, it runs the corresponding behavioral script BS_j on the designated host H_r at the scheduled session times in S_i.

For each tuple $(t_{2j-1}, t_{2j}) \in S_i$, the Sessions Executor starts a remote session on host H_r at time t_{2j-1}, runs BS_j until t_{2j}, and then terminates the session. This process repeats for all tuples in S_i until the final session (t_{2k_i-1}, t_{2k_i}).

Each department in the system can have a general behavioral script representing typical employee actions, or individual scripts can be assigned to users, configurable in the configuration file. During each execution block (Figure 2), a subset of behaviors from the script is randomly executed using a

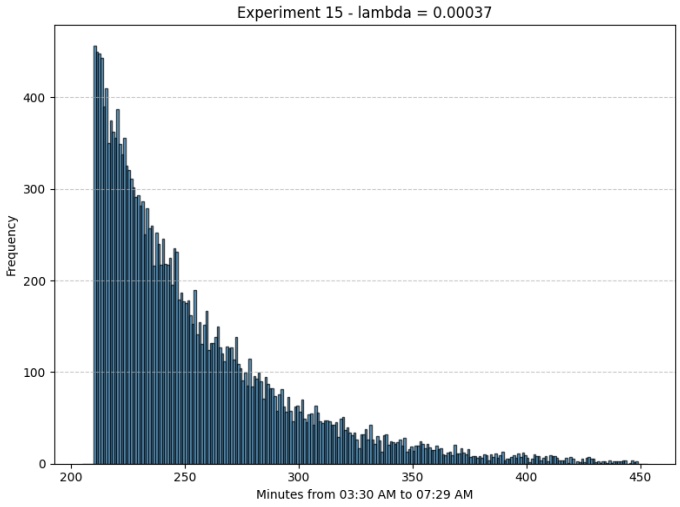

Fig. 3: Frequency distribution of $t_{\text{start\_abnormal\_early}}$ from 03:30 AM to 07:29 AM over 20,000 trials. The distribution follows an exponential distribution with a rate parameter $\lambda = 0.00037$, indicating higher frequencies of abnormal early start times occurring at earlier minutes and tapering off towards later minutes.

probabilistic approach. These behaviors include actions like browsing, downloading, running local programs, and accessing internal servers. The framework's scripts can be customized for specific enterprise use cases, enabling the simulation of various operational environments and realistic user behaviors.

This approach provides a flexible, scalable, and adaptable method for simulating user behavior in network environments, suitable for both small and large organizations. The source code for BDE is available on our GitHub [36], offering researchers access for dataset generation and other academic purposes.

D. Attack Engine (AE)

In this context, an *Attack Engine* refers to a method or framework that enables the automated execution of cyberattacks. For example, automating DDoS attacks can often be achieved by deploying specific scripts on the attacking hosts to initiate the attack. However, as we will discuss in this section, automating LM attacks presents unique challenges that are more complex and less straightforward than those associated with simpler scripted attacks.

1) LM Attacks: According to the MITRE ATT&CK framework [1], nine tactics qualify as LM techniques. These include *Exploitation of Remote Services*, *Internal Spearphishing*, *Lateral Tool Transfer*, *Remote Service Session Hijacking*, *Remote Services*, *Replication through Removable Media*, *Software Deployment Tools*, *Taint Shared Content*, and the *Use of Alternate Authentication Material*. In the LMDG dataset, multiple LM tactics from this list—such as Exploitation of Remote Services and the Use of Alternate Authentication Material—are employed, as discussed further below II-D5.

Each of these LM tactics encompasses various techniques. For instance, the "Use of Alternate Authentication Material" tactic can be executed through techniques like "Pass-the-Hash" or "Pass-the-Ticket" attacks. To clarify the complexity and unique nature of LM attacks compared to more straightforward attack types, we provide a detailed example of one of these attacks. This analysis highlights the operational challenges and automation complexities inherent in implementing these advanced tactics.

One of the attack scenarios demonstrated in the LMDG dataset involves a *pass-the-hash (PtH)* attack. An outline of the attack sequence is depicted in Figure 4. The scenario begins by assuming an attacker has obtained the local administrator credentials for domain controller DC2 in subnet 1, potentially through techniques like phishing. Using these credentials, the attacker initiates an SSH connection to an SSH server in subnet 7 (attack step 1) and subsequently connects to DC2 (attack step 2) via SSH using the same credentials. Once on DC2, the attacker downloads and executes *Mimikatz* to extract credential hashes from the LSASS process, including those from recent sessions. In this case, an enterprise administrator recently accessed DC2 (shown by the green arrow in Figure 4), allowing the attacker to retrieve the administrator's credentials. The attacker gains an elevated shell with the enterprise admin hash (attack step 3), enabling access to restricted directories on a file server in subnet 6 (attack step 4). This elevated access allows sensitive information to be exfiltrated from a folder accessible only to the enterprise administrator.

2) Challenges in Automating LM Attacks: As shown in the earlier attack example II-D1, elevated and reverse shells are common in LM attacks. Automating such steps introduces key challenges. For instance, automating the *Pass-the-Hash* (PtH) attack from subsection II-D1 and Figure 4 involves feasible steps like SSH-ing with stolen credentials and executing commands on the domain controller (e.g., running Mimikatz). However, automating PtH to obtain an elevated shell is difficult, as the resulting *cmd.exe* process runs under elevated credentials and is hard to access without knowing its properties (e.g., PID). Reverse shells add further complexity, especially when spawned on different hosts. If the NTLM hash must also be dynamically extracted mid-attack, automation becomes even harder. These issues highlight two main automation challenges: managing elevated/reverse shells and dynamically retrieving and reusing data from earlier attack steps.

3) A Candidate Solution: A solution to automating elevated and reverse shells in LM attacks, as discussed in section II-D2, is a client-server architecture,i.e., an orchestrator issuing commands to agents deployed on hosts.

In the attack scenario from section II-D1, an orchestrator in subnet 7 controls agent A_1 on the SSH server. Instead of SSH-ing from A_1 to DC2, a new agent A_2 is spawned on DC2, executing commands like running *Mimikatz*. For the PtH operation, the PtH command's `/run` parameter is modified to spawn an elevated agent A_3, completing the final attack step to exfiltrate data.

Figure 5 shows the process tree during a Pass-the-Hash (PtH) attack using a client-server architecture. Agent A_2,

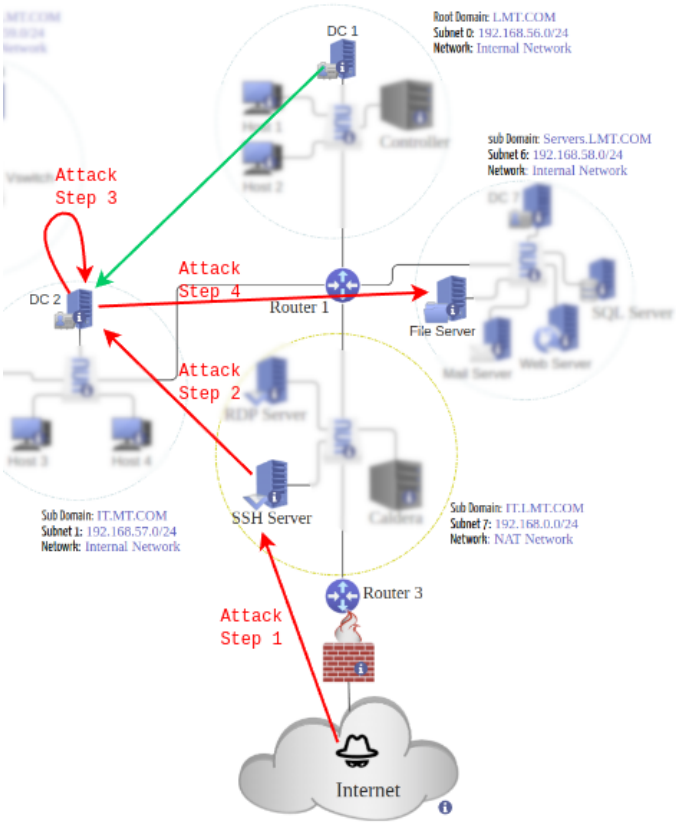

Fig. 4: First Attack Scenario in LMDG dataset which is Passing the Hash attack (PtH), providing a step-by-step visualization of the movement through network nodes.

represented by `splunkd_A2.exe`, is deployed on Domain Controller DC2 and performs the PtH attack. It then spawns Agent A_3 (`splunkd_A3.exe`) with elevated privileges. Agent A_3 communicates with the controller to continue the attack, accessing a restricted directory to retrieve data. This hierarchical structure visualizes how the client-server model manages privilege escalation, with attack steps chaining through communication between spawned agents. Tools like *CALDERA* support this feature [20].

4) CALDERA as an Attack Engine: Following the approach outlined in [20], we utilized Caldera to implement the client-server architecture discussed in subsection II-D3 to automate attack execution steps. As previously noted, Caldera, along with other tools referenced in [20], can be employed to facilitate this level of attack automation. We refer to such tools collectively as ”attack engines.”

Caldera™ [39] is an adversary emulation platform developed by MITRE for autonomous breach-and-attack simulations, manual red-team operations, and automated incident response. Based on the MITRE ATT&CK™ [1] framework, Caldera includes a core system consisting of the main framework code, an asynchronous command-and-control (C2) server, a REST API, and a web interface. It also supports plug-ins—separate repositories that extend the core functionality by adding agents, graphical interfaces, and collections of Tactics, Techniques, and Procedures (TTPs), enabling a flexible and

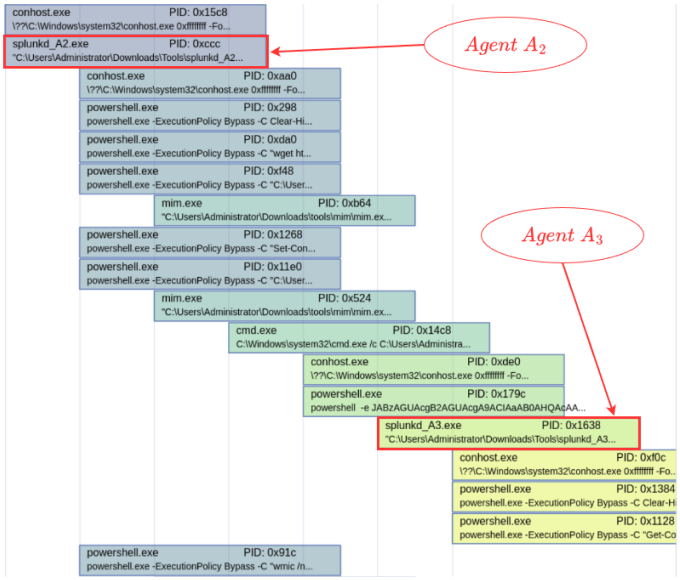

Fig. 5: Partial process tree illustrating the execution of the Pass-the-Hash (PtH) attack on DC2, as discussed in subsection II-D1.

comprehensive approach to adversary emulation.

5) LM Attacks in LMDG Dataset: The LMDG dataset contains seven attack scenarios that achieve LM using various tactics and techniques (see table I). Of these, three attacks were successful, while four were unsuccessful. We discuss possible reasons for each unsuccessful attack, considering that our setup includes Windows 10, Windows 11, and Windows Server 2022—the latest Windows versions with advanced security mechanisms [36]. This combination of successful and unsuccessful attacks is valuable for understanding attacker behavior, as many attacks tend to fail due to robust defenses, with only some achieving success.

Our dataset includes multiple versions of each attack, targeting different hosts and subnets. In some cases, attacks were executed repeatedly to enrich the dataset with diverse instances of attack records; we refer to this repetition of the same scenario, version pair, as a trial.

The attack steps depicted in the figures [36], e.g., figure 4, within the attack explanations represent LM hops, as defined in Section VI. All attack scenarios share the first two steps: initial access to the SSH server from outside the network using stolen credentials, followed by access to an additional internal machine. Beyond these initial steps, each attack scenario diverges in tactics and execution. More details about attacks execution are presented in III and our Github [36].

E. LMDG Labelling Engine (LE)

In cybersecurity datasets, labeling involves identifying and extracting records associated with attack activities from system logs and network traffic. The Labeling Engine serves as the component responsible for automating this extraction process. This subsection will examine the challenges of achieving accurate labeling and introduce our innovative labeling methodology.

Attack	Description
Passing the Hash	Use the hashed password of an enterprise admin to authenticate and gain unauthorized access.
Asreproastable	Exploit a user's AS-REP response to steal credentials and impersonate the administrator to steal data.
Pass the TGT	Dump LSASS memory to identify a domain admin, steal the TGT, inject it into memory, and steal data.
Attack Delegation	Perform AS-REP roasting, steal credentials, abuse group permissions (e.g., AddSelf), execute DC Sync to steal administration credentials, renew the TGT, and perform actions as the administrator.
Password Spray	Use password brute-forcing to open a zip file, perform a password spray attack to log in as an admin, abuse write permissions on a specific share to add a malicious file executed by a domain admin, and steal data.
Silver Ticket	Dump LSASS memory to identify a domain admin, use the stolen data to create a silver ticket, inject it into memory, and perform actions as the domain admin.
Golden Ticket	Dump LSASS memory to identify a domain admin, use the stolen data to create a golden ticket, inject it into memory, and perform actions as the domain admin.

TABLE I: Summary of Lateral Movement Attacks in LMDG Dataset

1) Challenges in Attack Data Labeling: A review of labeling techniques for cybersecurity dataset generation reveals three primary approaches: ***Injection Timing***, ***Behavioral Profiling***, and ***Network Security Tools*** [28].

The ***Injection Timing*** approach labels all logs or network traffic within the attack period as malicious, improving accuracy when combined with other methods [8], [14], [19], [22], [31]. However, it assumes no benign events occur during the attack, which leads to inaccuracies, particularly in complex attacks like LM, where benign activity may overlap with malicious actions.

The ***Behavioral Profiles*** method uses predefined profiles of malicious and benign behaviors for labeling [12], [43], [46]. While effective for identifying attack-specific characteristics, it fails in LM attacks, where legitimate users and hosts are exploited, rendering behavioral profiling insufficient.

The ***Network Security Tools*** approach uses data from security tools such as IDS, honeypots, and packet sniffers to label records [3], [43]. Despite its utility, this method faces accuracy issues, including false positives and negatives, due to tool limitations.

Given these challenges, there is a need for a more accurate labeling technique. We propose a novel methodology that addresses the shortcomings of existing approaches, improving accuracy, particularly for LM and advanced persistent threats (APTs), where traditional methods are less effective.

2) LMDG Labeling Engine: Our labeling methodology builds upon and extends the labeling approach introduced in [20], with specific enhancements and improvements outlined in the related work section V-A. We designate this approach as ***process tree labeling***, which can be considered an additional automatic labeling technique and, as we argue, the most accurate among those reviewed. The effectiveness of process tree labeling relies on the client-server architecture introduced in II-D3 and II-D4 for automating attack execution, a dependency explored in greater detail in [20].

Upon the completion of attack execution, the LMDG labeling engine, along with its input—a descriptive file containing metadata on attack steps—operates from the controller, depicted in Figure 1. The engine distributively performs the labeling task across each affected host, using the defined attack steps from the input file to extract the relevant subset of system logs and network connections associated with each attack stage on every affected host. The LMDG labeling engine completes this process in three primary phases: ***Attack Steps Forest Construction***, ***System Logs Labeling***, and ***Network Traffic Labeling***. Before detailing these stages, we will first discuss the input to this engine, namely the descriptive file containing attack steps metadata.

*a) **LMDG Labeling Engine Input**:* The input to the labeling engine consists of a set of hosts impacted by various attack steps, where each host includes a collection of malicious processes with specific attributes. Let H represent the set of all such hosts, i.e., $H = \{h_1, h_2, ..., h_n\}$, with each host $h_i \in H$ uniquely identified by a `HostName`. For each host h_i, let $P(h_i)$ denote the set of processes associated with the malicious agents deployed on that host during any attack step, i.e., $P(h_i) = \{p_1, p_2, ..., p_m\}$. Each process $p_j \in P(h_i)$ is described by the following tuple

$$p_j = (\pi,\ t_s,\ t_e,\ \sigma,\ \nu,\ \tau,\ \kappa,\ \phi)$$

In this tuple, π denotes the process identifier associated with a deployed Caldera agent, i.e., PID. t_s and t_e define the time window during which a particular attack step occurred (start time and end time). The specific step within the attack and the overarching scenario are identified by the κ and σ fields in the tuple, with ϕ indicating whether the step was completed successfully. Since an attack scenario can be executed across various hosts or subnets, multiple versions of the same scenario may exist. For example, in subsection II-D1, the pass-the-hash (PtH) attack can be executed on different subnets (e.g., subnet 4 instead of subnet 1). This versioning is captured by the ν field in the tuple. Additionally, we may execute the same scenario version multiple times; thus, the τ field is included to distinguish between these instances, offering clear differentiation across repeated executions or trials.

Each host $h_i \in H$ can be formally represented as a tuple containing its `HostName` and the set of associated processes, $P(h_i)$, for each Caldera agent deployed on that host. Formally, this is expressed as $h_i = (\texttt{HostName}_i, P(h_i))$. The overall input structure can then be denoted by Input $= \{h \mid h \in H\}$. This structured organization facilitates the grouping of processes by host, enabling efficient distribution of the labeling process and correlation of process executions with the various stages of distinct attack scenarios.

*b) **Attack Steps Forest Construction**:* The first stage in our labeling engine is the *Attack Steps Forest Construction*,

where we build a forest $\mathcal{F}$ of m process trees $\mathcal{T}_{p_j}$, one for each malicious process $p_j \in P(h_i) = \{p_1, \ldots, p_m\}$ on host h_i. Each tree is rooted at p_j's PID and includes all descendant processes within its execution window $[t_s, t_e]$. This temporal constraint ensures that each $\mathcal{T}_{p_j}$ captures causally relevant attack activity, providing the foundation for accurate step-level labeling.

Algorithm 1 Attack Steps Forest Construction

procedure ATTACKSTEPSFORESTCONSTRUCTION(H)
  $\mathcal{F} \leftarrow \emptyset$ ▷ Initialize the forest of attack steps
  for $h_i \in H$ **do**
    for $p_j \in P(h_i)$ **do**
      $\pi \leftarrow p_j.\pi, \quad t_s \leftarrow p_j.t_s, \quad t_e \leftarrow p_j.t_e$
      $\sigma \leftarrow p_j.\sigma, \quad \nu \leftarrow p_j.\nu, \quad \tau \leftarrow p_j.\tau$
      $\kappa \leftarrow p_j.\kappa, \quad \phi \leftarrow p_j.\phi$ ▷ Extract p_j's attributes
      $\mathcal{L} \leftarrow \emptyset$ ▷ Initialize a list for process IDs
      $\mathcal{T}_{p_j} \leftarrow$ GETPROCESSTREE$(\pi, t_s, t_e, \mathcal{L})$ ▷ Build process tree of process p_j
      $\mathcal{T}_{p_{j\,\text{meta}}} \leftarrow (\mathcal{T}_{p_j}, t_s, t_e, \sigma, \nu, \tau, \kappa, \phi)$
      $\mathcal{F} \leftarrow \mathcal{F} \cup \{\mathcal{T}_{p_{j\,\text{meta}}}\}$
    end for
  end for
  return $\mathcal{F}$
end procedure
procedure GETPROCESSTREE($\pi, t_s, t_e, \mathcal{L}$)
  $\mathcal{L} \leftarrow \mathcal{L} \cup \{\pi\}$
  $\mathcal{E} \leftarrow \{e \in \mathcal{E}_{h_i}^{4688} \mid e.\pi = \pi \ \wedge \ t_s \leq e.t \leq t_e\}$
  for $e \in \mathcal{E}$ **do**
    if $e.\pi \notin \mathcal{L}$ **then**
      $\mathcal{L} \leftarrow \mathcal{L} \cup \{e.\pi\}$
      GETPROCESSTREE$(e.\pi, t_s, t_e, \mathcal{L})$
    end if
  end for
  return $\mathcal{L}$
end procedure

The specifics of this step are outlined in Algorithm 1, where the set $\mathcal{E}_{h_i}^{4688}$ represents all process creation events recorded in the Windows Security log for host h_i.

An example of the output generated by Algorithm 1 is shown in Figure 6. This output corresponds to the example previously detailed in Subsection II-D1, which illustrates a Pass-the-Hash (PtH) attack scenario, as depicted in Figure 4.

In Figure 6, the example demonstrates two process trees rooted at the same process ID π of p_r. The first tree is constructed within the constrained time interval $[t_1, t_2]$, encompassing all subprocesses that occurred within this interval and represents attack step 3 from Figure 4. The second tree is built under the time constraint $[t_3, t_4]$, including all subprocesses within this later interval, and represents attack step 4 from the same figure 4.

We next move to the subsequent steps, System Logs Labeling and Network Traffic Labeling, which depend on the constructed attack steps forest $\mathcal{F}$.

*c) **System Logs Labeling**:* In this step, for each host h_i, we iterate over the set $\mathcal{L}_{h_i}$, which represents the collection of

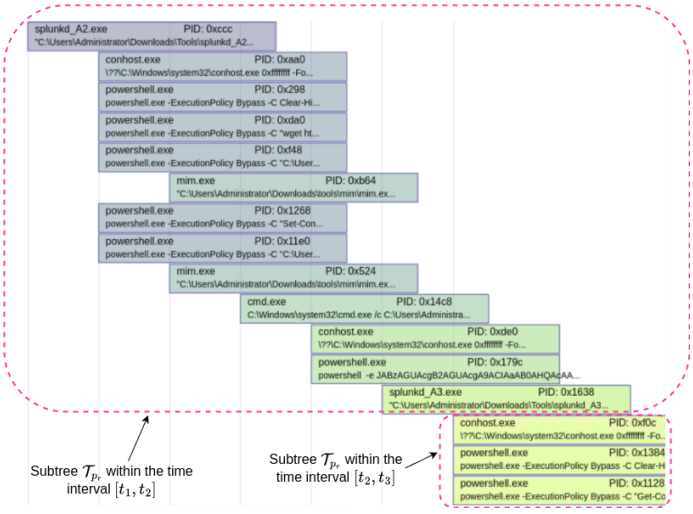

Fig. 6: Output of algorithm 1 showing two process trees rooted at the same malicious process p_r. The first tree, representing attack step 3, and the second tree, representing attack step 4 in Figure 4 explained in subsection II-D1.

all Windows event logs on that host. For each log $l \in \mathcal{L}_{h_i}$, we further iterate over the trees in the constructed forest $\mathcal{F}$ at host h_i, where each tree $\mathcal{T}_{p_j} \in \mathcal{F}$ corresponds to a specific attack step. The primary objective here is to examine whether the current log l contains any events with process IDs matching those within the current tree $\mathcal{T}_{p_j}$ that occurred within the specified time interval $[t_s, t_e]$. If such events are found, they are extracted and tagged with metadata of the current tree $\mathcal{T}_{p_j}$, including details like the attack scenario, version, step number, and step success status. This labeling process operates at the ***attack step level***, incorporating relevant MITRE ATT&CK tactics and techniques to contextualize each event within the broader attack framework.

*d) **Network Traffic Labeling.**:* In this step, we construct the set $\mathcal{E}_{h_i}^{5156} = \{e \mid e.\text{EventID} = 5156 \wedge e \in \mathcal{E}_{h_i}\}$, which represents the collection of Windows events with Event ID 5156, corresponding to the Windows Filtering Platform (WFP). The WFP monitors and filters network traffic on Windows systems, and $\mathcal{E}_{h_i}$ denotes the set of all Windows event logs at host h_i. Subsequently, we iterate over the trees in the constructed forest $\mathcal{F}$ and examine whether any process ID in the current tree $\mathcal{T}_{p_j}$ matches a process ID from the events in $\mathcal{E}_{h_i}^{5156}$. If a match is found, we filter the relevant events and label them with the metadata of the corresponding tree $\mathcal{T}_{p_j}$, including attack scenario, version, attack step, step success, and so on. Similar to the System Logs Labeling step, we also consider MITRE ATT&CK tactics and techniques for contextualizing the events within the attack framework.

The process of network flow labeling can be effectively performed using the packet capture (PCAP) files collected from each host, as mentioned in subsection II-B and the output of this labeling step.

Our "Process Tree Labeling" methodology provides superior accuracy in automated labeling by associating process activities with specific attack steps. By utilizing temporal and contextual information from system logs and network traffic,

it ensures precise event attribution within the attack lifecycle, enhancing labeling fidelity and reliability.

III. Dataset

The experimental environment consists of 25 VMs, including a Controller, Caldera server, domain controllers, application servers, hosts, and routers. While 22 user accounts were set up, only 11 credentials were used by the Benign Data Engine to generate benign data. Windows Event logs and PCAP files were collected from all Windows machines except the Controller and Caldera server. PCAP files were also captured from routers 1 and 2 for supplementary network data.

The dataset was generated over 25 days (October 10–November 3, 2024), with continuous benign data generation by the Benign Data Engine. Attacks occurred between October 23 and November 1, 2024, resulting in both benign and malicious data during this period. The dataset contains only benign data before October 23, 2024.

We present statistics on attacks within the LMDG dataset. Figure 7 shows the *Daily Distribution of Attack Steps*, illustrating the frequency of attack steps over time, with the total attack size in the dataset under 1%.

Figure 8 displays the *Timeline of Attack Step Occurrences by Scenario*, showing attack step timings via a scatter plot with distinct color coding for different scenarios.

Figure 9 presents the *Frequency Distribution of Scenario and Version Pairs*, comparing attack frequencies across scenario versions. The full execution timeline is available on our GitHub repository [36].

The compressed dataset size is 253 GB (excluding router data) and 527 GB (including router data), with router 1's PCAP at 201 GB and router 2's at 72 GB. The total uncompressed dataset is 944 GB, comprising 900.93 GB of PCAP files and 43.38 GB of system logs (Table II).

Like the LANL 2015 dataset, this dataset enables detailed extraction of authentication features and patterns, supporting research in intrusion detection, behavioral analysis, and user activity monitoring. Its rich event data and metadata provide a solid foundation for diverse cybersecurity research tasks.

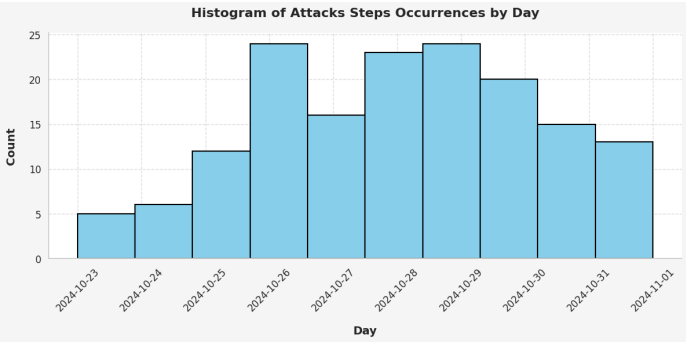

Fig. 7: **Daily Distribution of Attack Steps**: This histogram visualizes the frequency of attack steps executed over time, with each bar representing the count of attack steps occurring on a specific day. The x-axis denotes individual days, while the y-axis represents the number of occurrences.

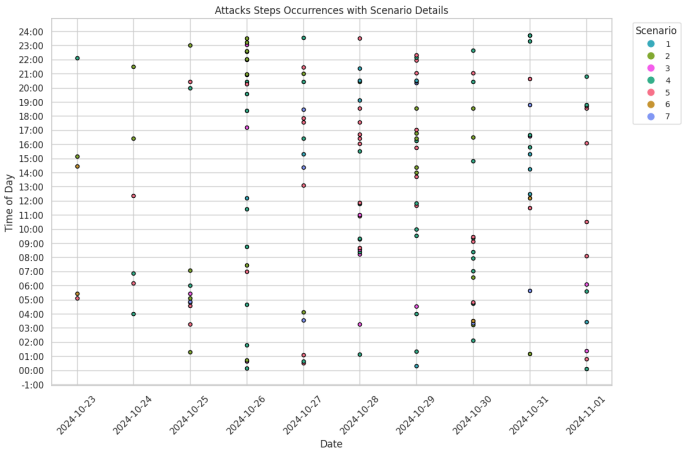

Fig. 8: **Timeline of Attack Step Occurrences by Scenario**: Thacross various days, with each point representing the occurrence of an attack step on a specific day and time. The x-axis indicates the occurrence dates, while the y-axis represents the time of day to highlight daily distribution patterns. Each scenario is color-coded with a distinct hue, allowing for quick differentiation of scenarios.

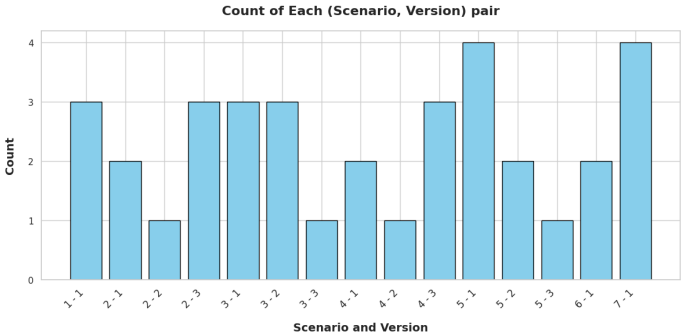

Fig. 9: **Frequency Distribution of Scenario and Version pairs**: This bar plot displays the count of occurrences for each distinct (Scenario, Version) pair. The x-axis represents individual combinations of scenarios and their respective versions. The y-axis shows the count of occurrences.

is scatter plot illustrates the timing of attack steps

TABLE II: Dataset Statistics

Statistic	PCAP Size (GB)	Log Size (GB)
Total Size	900.93	43.38
Average Size	37.54	2.17
Minimum Size	0.51	0.41
Maximum Size	451.00	4.97

IV. Qualitative Analysis

Dataset quality depends on several phases: testbed design, benign data generation, log collection, attack execution, and labeling. Enhancing these phases improves dataset robustness.

Our testbed (Section II-B) simulates a realistic enterprise network using virtualization for scalable and detailed emulation. The Benign Data Engine (BDE) (Section II-E2, Figure 2) generates realistic benign data based on behavioral scripts. These scripts, modeled on departmental roles with randomiza-

tion, replicate typical employee activities such as logins, web browsing, service requests, and program execution.

Our dataset includes comprehensive system logs and network traffic data, with labeling applied to both, ensuring complete activity records for detailed behavior analysis.

Existing LM datasets have limitations such as few LM instances, outdated patterns, limited techniques, short timeframes, and minimal hops. In contrast, our LM attacks (Section II-D5) address these issues with diverse techniques, recent patterns, a 10-day execution window, and up to 7 hops across hosts, users, and subnets. Tools like CALDERA further improve attack design flexibility.

Our automated *process tree labeling* (Section II-E) achieves superior accuracy, effectively tracking process hierarchies critical for LM and APT attacks.

The LMDG framework is designed to support the generation of high-quality datasets through its various integrated components. The LMDG dataset serves as an exemplar of this capability, demonstrating the framework's effectiveness in producing datasets that are comprehensive, well-structured, and suitable for advanced research and analysis.

V. Related Work

A. LADEMU Framework

LADEMU [20] uses virtualization tools (e.g., VirtualBox, VMware) for ***testbed infrastructure***, similar to LMDG. For ***dataset collection***, it captures network traffic (pcap) and Sysmon log, whereas LMDG expands coverage by collecting all Windows event logs and labeling them comprehensively. In ***benign data generation***, LADEMU employs GHOST, while LMDG offers greater flexibility through its Benign Data Engine (BDE) (II-C), which separates session scheduling from behavior execution. Both frameworks use Caldera for ***attack execution*** II-D. For ***labeling***, LADEMU builds malicious process trees from Sysmon logs, a method LMDG also adopts II-E with four key improvements. LADEMU constructs malicious process trees rooted at the Caldera agent and labels all benign process events during interaction intervals $[t_1, t_2]$ as malicious [31]. This can lead to mislabeling, especially in cases like code injection, where benign processes may act maliciously over extended periods. In contrast, LMDG labels only events linked to malicious trees, avoiding such inaccuracies. While LADEMU uses Caldera's start and finish times to bound trees [31], this can miss delayed malicious activity. LMDG extends the end time beyond Caldera's to ensure full coverage. LADEMU includes C&C signals in logs [31], reducing realism. LMDG filters these out to better mimic real-world scenarios. Finally, LMDG links labels to attack steps and scenarios [20], enabling more context-aware datasets and improving multi-step attack detection.

B. AIT Framework

The AIT framework by Landauer et al. [29]–[31] offers a model-driven approach [17] to cybersecurity dataset generation, emphasizing *automated testbed creation* [29], [31] and a *labeling methodology* [29]. The Kyoush platform, built with Terraform, Ansible, and OpenStack [31], enables reusable, ***automated testbed deployment*** but requires significant setup effort. Their ***labeling*** combines injection timing [21], [28] with manual query-based log inspection for each attack step [29], limiting scalability. While ***attack scripts*** were automated, the framework lacks support for complex attacks like LM II-D. ***Data collection*** targets Linux logs, and ***benign behavior*** is simulated via a User State Machine.

C. CREME Framework

CREME, introduced by Bui et al. [12], generates labeled intrusion detection datasets and evaluates dataset quality. It employs virtualization for ***testbed infrastructure*** and ***collects data*** via tcpdump, rsyslog, and Atop, but supports only Linux/Unix systems. For ***benign data generation***, its "Reproduction Module" runs unspecified benign programs; it also executes five ***attack*** types, though support for complex LM attacks is unclear II-D. ***Labeling*** uses Behavioral Profiles [21], [28], tagging all traffic from attack machines as malicious, which can not handle LM attacks II-E1.

VI. Discussion

Our review of LM detection reveals a need for a comprehensive definition of LM . While MITRE ATT&CK [1] defines it broadly as techniques for accessing and controlling remote systems, this vague description limits the effectiveness of detection models, highlighting the need for a more precise and actionable definition.

We define two key types of adversary progression: *horizontal progression* and *vertical progression*. *Horizontal progression* involves gaining independent access to multiple hosts without interdependence, which does not qualify as LM . In contrast, *vertical progression* describes interconnected access, where controlling one system enables access to others. ***We define LM as vertical progression across hosts, accounts, or privileges, where one access leads to another***. This includes movement between hosts, accounts with elevated privileges, and privilege escalation. This refined definition is essential for creating effective detection models.

In the cloud environment, LM follows a similar concept with modifications. *Identities* (user, application, and service accounts) correspond to *accounts*, requiring authentication to access resources, while *permissions* or *policies* align with *privileges*, defining access levels. A key distinction in the cloud is the *services layer*, which includes resources like AWS EC2 and S3, providing computing and storage. Thus, *cloud LM involves vertical progression across identities, permissions/policies, services, and resources*. For example, an attack detailed by Microsoft Threat Intelligence [25] involved exploiting SQL injection to access an Azure database server and using the Instance Metadata Service (IMDS) to obtain further access to cloud resources.

Regarding the threat model in the LMDG dataset, it emulates realistic APT-like scenarios where adversaries perform stealthy, persistent attacks over an extended period. These

scenarios include initial access, privilege escalation, and multi-hop LM across hosts and network subnets, reflecting the sophisticated behaviors of modern attackers targeting enterprise networks. By leveraging the CALDERA platform for attack emulation, the framework enables flexible attack design, supporting a variety of LM techniques and bypassing typical security defenses. To enhance realism, these attacks occur within a backdrop of benign user activities generated by the Benign Data Engine (BDE), providing a nuanced environment for distinguishing between normal and malicious behavior. This threat model thus offers a robust foundation for evaluating detection and response mechanisms against complex and dynamic cyber threats.

VII. Conclusions and Future Work

In this work, we have comprehensively examined current cybersecurity benchmark datasets with a specific focus on evaluating the presence and characteristics of LM attacks. Our analysis, the first of its kind, assessed LM datasets across multiple dimensions, including the quantity and variety of LM techniques, attack duration, number of movement hops, data sources (e.g., authentication logs, network flows), labeling methodologies, and testbed configurations. This investigation has highlighted gaps and challenges within existing datasets, providing insight into the strengths and limitations of current approaches to LM detection.

We developed a benchmark dataset focused explicitly on LM attacks to address the identified limitations. This dataset, designed to overcome many existing issues in LM datasets, provides a valuable resource for the research community, facilitating the training and evaluation of more effective LM detection models. Our qualitative dataset analysis demonstrates its applicability for various LM scenarios. It ensures that the diversity and complexity of attacks are suitable for testing advanced detection techniques.

Additionally, we introduced the LM Dataset Generator (LMDG) framework, a reproducible toolset for generating high-quality LM and APT datasets. The LMDG framework automates benign data generation, attack execution, and—crucially—the labeling of attack-related events in system and network logs. Recognizing the challenges posed by automatic labeling in LM scenarios, where benign hosts may perform malicious actions, we proposed a novel technique, *process tree labeling*. This method offers improved precision and accuracy over existing techniques such as injection timing, behavior profiles, and network security tools. Overall, the contributions of this work enhance the landscape of LM dataset generation and analysis, supporting further advancements in cybersecurity research and LM detection capabilities.

Several limitations of our framework warrant consideration. First, using virtualization to construct testbeds necessitates extensive domain expertise, making the process time-consuming and highly case-dependent, as discussed in more detail in [31]. This requirement for specialized knowledge may hinder the framework's scalability and accessibility. Second, the client-server architecture employed in attack automation, as outlined in Sections II-D3 and II-D4, introduces traffic and log accuracy challenges. Specifically, the traffic generated by client-server communication must be filtered to avoid contaminating the dataset with automation-related signals, ensuring that the resulting data remains realistic and reflective of actual attack behaviors. Finally, our proposed labeling methodology, process tree labeling, the most accurate automatic labeling technique, is inherently tied to the client-server automation model. This dependency arises from the need to identify the process IDs of deployed agents, creating a coupling between labeling and attack automation. This coupling is discussed in more detail in [20] and may limit the applicability of our labeling approach in environments where such client-server structures are not feasible.

While this study includes a qualitative analysis of our dataset IV and comparisons with existing datasets in the literature, further work is needed to incorporate quantitative analysis methods. A systematic review of current quantitative assessment techniques used in cybersecurity datasets will enable us to apply rigorous, data-driven evaluation metrics to our dataset, enhancing its reliability and usability. In addition, future efforts may focus on producing a more comprehensive dataset that encompasses the full spectrum of Advanced Persistent Threat (APT) attack stages rather than concentrating solely on LM . Such a dataset would capture all phases of APT attacks, offering a richer resource for developing and benchmarking holistic detection models that address the complete lifecycle of sophisticated attack vectors. This extension will advance research into multi-stage threat detection, providing excellent value for the cybersecurity community.

References

[1] MITRE ATT&CK®. [Online]. Available: https://attack.mitre.org/, 2024.

[2] Adel Alshamrani, Sowmya Myneni, Ankur Chowdhary, and Dijiang Huang. A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities. *IEEE Communications Surveys & Tutorials*, 21(2):1851–1877, 2019.

[3] Francisco J Aparicio-Navarro, Konstantinos G Kyriakopoulos, and David J Parish. Automatic dataset labelling and feature selection for intrusion detection systems. In *2014 IEEE Military Communications Conference*, pages 46–51. IEEE, 2014.

[4] Giovanni Apruzzese, Fabio Pierazzi, Michele Colajanni, and Mirco Marchetti. Detection and threat prioritization of pivoting attacks in large networks. *IEEE Transactions on Emerging Topics in Computing*, 8(2):404–415, 2020.

[5] Giovanni Apruzzese, Fabio Pierazzi, Michele Colajanni, and Mirco Marchetti. Detection and threat prioritization of pivoting attacks in large networks. *IEEE Transactions on Emerging Topics in Computing*, 8:404–415, 2020.

[6] Lora Aroyo, Matthew Lease, Praveen Paritosh, and Mike Schaekermann. Data excellence for ai: why should you care? *Interactions*, 29(2):66–69, feb 2022.

[7] Tim Bai, Haibo Bian, Mohammad A. Salahuddin, Abbas Abou Daya, Noura Limam, and Raouf Boutaba. Rdp-based lateral movement detection using machine learning. *Computer Communications*, 165:9–19, 2021.

[8] Monowar H Bhuyan, Dhruba K Bhattacharyya, and Jugal K Kalita. Towards generating real-life datasets for network intrusion detection. *Int. J. Netw. Secur.*, 17(6):683–701, 2015.

[9] Haibo Bian, Tim Bai, Mohammad A. Salahuddin, Noura Limam, Abbas Abou Daya, and Raouf Boutaba. Uncovering lateral movement using authentication logs. *IEEE Transactions on Network and Service Management*, 18(1):1049–1063, 2021.

[10] Atul Bohara, Mohammad A. Noureddine, Ahmed Fawaz, and William H. Sanders. An unsupervised multi-detector approach for identifying malicious lateral movement. In *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*, pages 224–233, 2017.

[11] Benjamin Bowman, Craig Laprade, Yuede Ji, and H. Howie Huang. Detecting lateral movement in enterprise computer networks with unsupervised graph AI. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, pages 257–268, San Sebastian, October 2020. USENIX Association.

[12] Huu-Khoi Bui, Ying-Dar Lin, Ren-Hung Hwang, Po-Ching Lin, Van-Linh Nguyen, and Yuan-Cheng Lai. Creme: A toolchain of automatic dataset collection for machine learning in intrusion detection. *Journal of Network and Computer Applications*, 193:103212, 2021.

[13] José Camacho, Katarzyna Wasielewska, Pablo Espinosa, and Marta Fuentes-García. Quality in / quality out: Data quality more relevant than model choice in anomaly detection with the ugr'16. In *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, pages 1–5, 2023.

[14] Gideon Creech and Jiankun Hu. Generation of a new ids test dataset: Time to retire the kdd collection. In *2013 IEEE wireless communications and networking conference (WCNC)*, pages 4487–4492. IEEE, 2013.

[15] Gustavo de Carvalho Bertoli, Lourenço Alves Pereira Júnior, Filipe Alves Neto Verri, Aldri Luiz dos Santos, and Osamu Saotome. Bridging the gap to real-world for network intrusion detection systems with data-centric approach. *CoRR*, abs/2110.13655, 2021.

[16] Emily Denton, Alex Hanna, Razvan Amironesei, Andrew Smart, Hilary Nicole, and Morgan Klaus Scheuerman. Bringing the people back in: Contesting benchmark machine learning datasets. *CoRR*, abs/2007.07399, 2020.

[17] Fermin Galan, David Fernandez, Jorge E. Lopez de Vergara, and Ramon Casellas. Using a model-driven architecture for technology-independent scenario configuration in networking testbeds. *IEEE Communications Magazine*, 48(12):132–141, 2010.

[18] Chenquan Gan, Jiabin Lin, Da-Wen Huang, Qingyi Zhu, and Liang Tian. Advanced persistent threats and their defense methods in industrial internet of things: A survey. *Mathematics*, 11(14), 2023.

[19] Sebastian Garcia, Martin Grill, Jan Stiborek, and Alejandro Zunino. An empirical comparison of botnet detection methods. *computers & security*, 45:100–123, 2014.

[20] Julie Gjerstad, Fikret Kadiric, Gudmund Grov, Espen Hammer Kjellstadli, and Markus Leira Asprusten. Lademu: a modular & continuous approach for generating labelled apt datasets from emulations. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 2610–2619. IEEE, 2022.

[21] Jorge Luis Guerra, Carlos Catania, and Eduardo Veas. Datasets are not enough: Challenges in labeling network traffic. *Computers & Security*, 120:102810, 2022.

[22] Waqas Haider, Jiankun Hu, Jill Slay, Benjamin P Turnbull, and Yi Xie. Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling. *Journal of Network and Computer Applications*, 87:185–192, 2017.

[23] Xueyuan Han, Thomas Pasquier, Adam Bates, James Mickens, and Margo Seltzer. Unicorn: Runtime provenance-based detector for advanced persistent threats. In *Proceedings 2020 Network and Distributed System Security Symposium*, NDSS 2020. Internet Society, 2020.

[24] Grant Ho, Mayank Dhiman, Devdatta Akhawe, Vern Paxson, Stefan Savage, Geoffrey M. Voelker, and David Wagner. Hopper: Modeling and detecting lateral movement. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 3093–3110. USENIX Association, August 2021.

[25] Microsoft Threat Intelligence. Defending new vectors: Threat actors attempt sql server to cloud lateral movement. October 3, 2023.

[26] A. Kenyon, L. Deka, and D. Elizondo. Are public intrusion datasets fit for purpose characterising the state of the art in intrusion event datasets. *Computers & Security*, 99:102022, 2020.

[27] Sean Michael Kerner. Crowdstrike outage explained: What caused it and what's next. *TechTarget*, Oct 2024. A CrowdStrike update caused a massive IT outage, crashing millions of Windows systems. Critical services and business operations were disrupted, revealing tech reliance risks.

[28] Meejoung Kim and Inkyu Lee. Human-guided auto-labeling for network traffic data: The gelm approach. *Neural networks*, 152:510–526, 2022.

[29] Max Landauer, Maximilian Frank, Florian Skopik, Wolfgang Hotwagner, Markus Wurzenberger, and Andreas Rauber. A framework for automatic labeling of log datasets from model-driven testbeds for hids evaluation. In *Proceedings of the 2022 ACM Workshop on Secure and Trustworthy Cyber-Physical Systems*, pages 77–86, 2022.

[30] Max Landauer, Florian Skopik, Maximilian Frank, Wolfgang Hotwagner, Markus Wurzenberger, and Andreas Rauber. Maintainable log datasets for evaluation of intrusion detection systems. *IEEE Trans. Dependable Secur. Comput.*, 20(4):3466–3482, jul 2023.

[31] Max Landauer, Florian Skopik, Markus Wurzenberger, Wolfgang Hotwagner, and Andreas Rauber. Have it your way: Generating customized log datasets with a model-driven simulation testbed. *IEEE Transactions on Reliability*, 70(1):402–415, 2020.

[32] Fucheng Liu, Yu Wen, Yanna Wu, Shuangshuang Liang, Xihe Jiang, and Dan Meng. Mltracer: Malicious logins detection system via graph neural network. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 715–726, 2020.

[33] Qingyun Liu, Jack W. Stokes, Rob Mead, Tim Burrell, Ian Hellen, John Lambert, Andrey Marochko, and Weidong Cui. Latte: Large-scale lateral movement detection. In *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*, pages 1–6, 2018.

[34] Xiaohan Ma, Chen Li, and Bibo Tu. An unsupervised approach for detecting lateral movement logins based on knowledge graph. In *2022 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 700–707, 2022.

[35] Anas Mabrouk. Lateral movement attacks datasets: Benchmarking, challenges, and solutions. M.sc. thesis, University of Windsor, Windsor, Ontario, Canada, 2024. Available from the University of Windsor Digital Repository.

[36] Anas Mabrouk, Mohammad Mamun, Sherif Saad, and Mohamed Hatem. Lateral movement dataset generation (LMDG): Dataset and documentation, 2025. Available at: https://github.com/WASPLab/LMTrace.

[37] Mark Mazumder, Colby R. Banbury, Xiaozhe Yao, Bojan Karlavs, William Gaviria Rojas, Sudnya Diamos, Gregory Frederick Diamos, Lynn He, Douwe Kiela, David Jurado, David Kanter, Rafael Mosquera, Juan Ciro, Lora Aroyo, Bilge Acun, Sabri Eyuboglu, Amirata Ghorbani, Emmett D. Goodman, Tariq Kane, Christine R. Kirkpatrick, Tzu-Sheng Kuo, Jonas W. Mueller, Tristan Thrush, Joaquin Vanschoren, Margaret J. Warren, Adina Williams, Serena Yeung, Newsha Ardalani, Praveen K. Paritosh, Ce Zhang, James Y. Zou, Carole-Jean Wu, Cody Coleman, Andrew Y. Ng, Peter Mattson, and Vijay Janapa Reddi. Dataperf: Benchmarks for data-centric ai development. *ArXiv*, abs/2207.10062, 2022.

[38] Grant McDonald, Pavlos Papadopoulos, Nikolaos Pitropakis, Jawad Ahmad, and William J Buchanan. Ransomware: Analysing the impact on windows active directory domain services. *Sensors*, 22(3):953, 2022.

[39] MITRE Corporation. *Caldera Documentation*, 2024. Accessed: 2024-10-31.

[40] Basem Ibrahim Mokhtar, Anca D Jurcut, Mahmoud Said ElSayed, and Marianne A Azer. Active directory attacks—steps, types, and signatures. *Electronics*, 11(16):2629, 2022.

[41] Kate O'Flaherty. CrowdStrike Reveals What Happened, Why—And What's Changed. *Forbes*, 2024. Senior Contributor; Kate O'Flaherty is a cybersecurity and privacy journalist.

[42] Emilie Purvine, John R. Johnson, and Chaomei Lo. A graph-based impact metric for mitigating lateral movement cyber attacks. In *Proceedings of the 2016 ACM Workshop on Automated Decision Making for Active Cyber Defense*, SafeConfig '16, page 45–52, New York, NY, USA, 2016. Association for Computing Machinery.

[43] Markus Ring, Sarah Wunderlich, Dominik Grüdl, Dieter Landes, and Andreas Hotho. Flow-based benchmark data sets for intrusion detection. In *Proceedings of the 16th European conference on cyber warfare and security. ACPI*, pages 361–369, 2017.

[44] Nithya Sambasivan, Shivani Kapania, Hannah Highfill, Diana Akrong, Praveen Kumar Paritosh, and Lora Mois Aroyo. "everyone wants to do the model work, not the data work": Data cascades in high-stakes ai. 2021.

[45] Amit Sharma, Brij B Gupta, Awadhesh Singh, and V. Saraswat. Advanced persistent threats (apt): evolution, anatomy, attribution and countermeasures. *Journal of Ambient Intelligence and Humanized Computing*, 14:1–27, 05 2023.

[46] Hadi Shiravi, Ali Shiravi, and Ali A Ghorbani. A survey of visualization systems for network security. *IEEE Transactions on visualization and computer graphics*, 18(8):1313–1329, 2011.

[47] Christos Smiliotopoulos, Georgios Kambourakis, and Konstantia Barmpatsalou. On the detection of lateral movement through supervised machine learning and an open-source tool to create turnkey datasets from sysmon logs. *International Journal of Information Security*, 22:1–27, 07 2023.

[48] Branka Stojanović, Katharina Hofer-Schmitz, and Ulrike Kleb. Apt datasets and attack modeling for automated detection methods: A review. *Computers & Security*, 92:101734, 2020.

[49] Xiaoqing Sun and Jiahai Yang. Hetglm: Lateral movement detection by discovering anomalous links with heterogeneous graph neural network. In *2022 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, pages 404–411, 2022.

[50] Ngan Tran, Haihua Chen, Jay Bhuyan, and Junhua Ding. Data curation and quality evaluation for machine learning-based cyber intrusion detection. *IEEE Access*, 10:121900–121923, 2022.

[51] Martin Ussath, David Jaeger, Feng Cheng, and Christoph Meinel. Advanced persistent threats: Behind the scenes. In *2016 Annual Conference on Information Science and Systems (CISS)*, pages 181–186, 2016.