Accurate and Scalable Detection and Investigation of Cyber Persistence Threats

Qi Liu, Muhammad Shoaib, Mati Ur Rehman, Kaibin Bao, Veit Hagenmeyer, Wajih Ul Hassan

***Abstract*—In Advanced Persistent Threat (APT) attacks, achieving stealthy persistence within target systems is often crucial for an attacker's success. This persistence allows adversaries to maintain prolonged access, often evading detection mechanisms. Recognizing its pivotal role in the APT lifecycle, this paper introduces Cyber Persistence Detector (CPD), a novel system dedicated to detecting cyber persistence through provenance analytics. CPD is founded on the insight that persistent operations typically manifest in two phases: the "persistence setup" and the subsequent "persistence execution". By causally relating these phases, we enhance our ability to detect persistent threats. First, CPD discerns setups signaling an impending persistent threat and then traces processes linked to remote connections to identify persistence execution activities. A key feature of our system is the introduction of *pseudo-dependency edges* (pseudo-edges), which effectively connect these disjoint phases using data provenance analysis, and *expert-guided edges*, which enable faster tracing and reduced log size. These edges empower us to detect persistence threats accurately and efficiently. Moreover, we propose a novel alert triage algorithm that further reduces false positives associated with persistence threats. Evaluations conducted on well-known datasets demonstrate that our system reduces the average false positive rate by 93% compared to state-of-the-art methods.**

***Index Terms*—Advanced Persistence Threat detection, data provenance analysis.**

I. Introduction

Advanced Persistent Threat (APT) attacks are increasingly leveraging Living-Off-the-Land Binaries (LOLBins), shifting the strategic focus from traditional malware to more nuanced persistence techniques. According to MITRE [1], persistence techniques are defined as methods that adversaries use to keep access to systems across restarts, changed credentials, or other interruptions that could cut off their access. These techniques typically involve the installation of malicious software or the manipulation of legitimate scripts and tasks to ensure continuous, unauthorized remote access. Techniques such as reverse shells, SSH, Powershell Remoting, or other executables are often used for establishing and maintaining remote connections. Notably, persistence techniques were a key feature in nearly 75% of cyberattacks in 2022 [2].

For instance, the Sandworm APT group used webshell persistence in multistage attacks [3]–[5]. The SolarWinds attack shows how persistence is crucial in APT campaigns, using scheduled tasks for this purpose [1], [6]–[8]. APT attackers often use a "low and slow" strategy, breaking their actions into phases with waiting periods to avoid detection. They gain initial access, establish persistence, disconnect to evade detection, and later reconnect for further malicious activities. This segmentation and strategic pausing are trademarks of the most stealthy APT attacks, as illustrated in Figure 1.

In addressing the complexities of APT attacks, Provenance-based Intrusion Detection Systems (PIDS) [9]–[20] have become essential by transforming audit logs into provenance graphs, providing causal relationships among system entities, such as processes and network sockets. In contrast, rule-based detection systems, such as Elastic [21], Google Chronicle [22], and Sigma [23], which match audit logs against a predefined set of signatures, are industry standards for their capability to identify persistence threats.

Persistence techniques, as defined by MITRE [1], often involve the misuse of "sensitive" system functionalities, such as Registry run keys [24]. Current threat detection systems generate persistence attack alerts whenever these functionalities are accessed, irrespective of whether they are being misused by an attacker or legitimately used by a normal user. This approach fails to assess the consequences of the use of system functionalities, which might only become apparent later. Consequently, this leads to a high number of false positives; for example, a normal user adding entries in the Registry run keys or startup folder to launch programs upon log-on would trigger an alert, even though the action is benign. Conversely, if an attacker sets a Registry run key to initiate a command and control (C2) agent that connects back post-reboot, existing systems would identify the key's creation but might not link it to subsequent C2 activities due to the delay in their occurrence and the lack of a comprehensive context check necessary for accurate persistence detection.

A. Limitations of Existing PIDS

Anomaly-based or learning-based PIDS [9], [17]–[20], [25]–[27] aim to detect novel attacks with less prior knowledge. These PIDS model benign behavior from provenance graphs, and detect deviations from the modeled normal behavior. However, the assumption that attack-related activities are anomalous is not always true. Besides, they require

This work was supported by funding of the Helmholtz Association (HGF) through the Energy System Design (ESD) program. We also acknowledge support by the Karlsruhe House of Young Scientists (KHYS) for the research stay of Qi Liu at University of Virginia.

Qi Liu, Kaibin Bao, and Veit Hagenmeyer are with Institute for Automation and Applied Informatics, Karlsruhe Institute of Technology (KIT), Eggenstein-Leopoldshafen 76344, Germany (e-mail: qi.liu@kit.edu; kaibin.bao@kit.edu; veit.hagenmeyer@kit.edu).

Muhammad Shoaib, Mati Ur Rehman, and Wajih Ul Hassan are with the School of Engineering & Applied Science, University of Virginia, Charlottesville, VA 22904-4740, USA (e-mail: mshoaib@virginia.edu; wkw9be@virginia.edu; hassan@virginia.edu).

representative training data which are not always available, and the training phase slows down the detection process. Last, learning-based PIDS are inherently more susceptible to concept drift and more vulnerable to evasion attacks [28], [29]. Our evaluation in Section VI shows that state-of-the-art learning-based PIDS [17], [18] are not only slower but also less accurate in persistence detection, due to failure to learn persistence attacks' semantics.

Existing heuristic-based PIDS also struggle with semantic understanding of persistence attacks, leading to significant challenges in connecting the dots across fragmented APT attack stages. This often results in incomplete and disconnected provenance (attack) graph reconstructions. Consider the attack scenario depicted in Figure 1; heuristic-based PIDS, such as Holmes [13] and RapSheet [15], often fail to piece together the full scope of an attack involving persistence. Instead, they produce isolated graphs, each representing only fragments of the APT attack. These systems triage attack graphs based on the number of APT stages, such as lateral movement and privilege escalation, contained within the graphs. If an attacker manages to fragment the attack graph into smaller, disconnected graphs, each segment inherently receives a lower severity score and is subsequently ranked lower for detection and investigation. Moreover, these heuristic-based PIDS require benign training data to quantify severity scores and filter false alarms.

B. Limitations of Rule-based Persistence Detectors

Current rule-based persistence detection systems, including popular solutions like Elastic [21] and Chronicle [22], are plagued by a high rate of false positives (FPs). These systems typically analyze persistence techniques in isolation, neglecting the broader context of an attack. They often generate alerts for system activities that appear suspicious but are actually benign, leading to numerous false alarms. In an effort to mitigate these false positives, these systems may overly relax their detection rules. For instance, our evaluation in Section VI revealed that activities from programs in standard directories are automatically deemed benign without further scrutiny. This approach has inadvertently resulted in a significant increase in false negatives (FNs).

The narrow detection strategy has tangible consequences in Security Operations Centers (SOCs), where analysts spend roughly 30 minutes on each alert [30], [31], but with up to 99% of these turning out to be FPs [32], leading to alert fatigue. To manage the deluge of alerts, thresholds are often set or high-volume alert rules are disabled, resulting in over two-thirds of alerts being disregarded [33], [34]. This underscores the dire need for a system that can automatically reduce alert numbers.

C. Our Approach and Contributions

To address the issues with current persistence threat detection, we introduce Cyber Persistence Detector (CPD), a novel system optimized for quick and accurate identification of persistence threats in enterprise networks. Our approach

- ⋆ avoids optimistic assumptions about persistence behavior,
- ⋆ avoids the need for any training data,
- ⋆ generates few FPs and FNs,

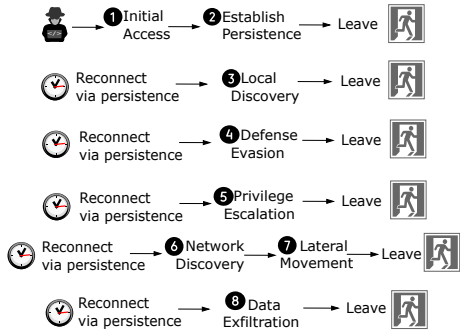

Fig. 1: Stealthiness by persistence

- ⋆ triages persistence-related threat alerts, and
- ⋆ generates accurate graphs for quick incident response.

CPD is rooted in a thorough analysis of the MITRE ATT&CK framework [35] which is recognized as the most comprehensive and widely referenced directory of persistence threats. We discovered that effective persistence attacks always consist of two phases: the persistence setup (e.g., creating a Registry run key) and the persistence execution (e.g., a remote connection initiated by that key). *Persistence setup serves solely as preparation, whereas persistence execution exhibits attackers' true motives*. Recognizing this two-phase process is crucial, yet it is overlooked by existing detection systems.

Leveraging the above key insight, CPD introduces a novel concept of *pseudo-dependency edges* (pseudo-edges) to connect persistence setup and persistence execution activities within system logs, creating a comprehensive provenance graph. CPD starts by recording every persistence setup activity in system logs, and then specifically checks if there is a subsequent remote connection, i.e., potential persistence execution, that can be traced back to the persistence setup activity. If not, it does not raise an alert, significantly reducing false positives. If yes, it creates a pseudo-edge, and then utilizes even more contextual indicators, detailed in Section IV-C, before deciding if it is a persistence attack. The creation of a pseudo-edge involves tracing processes in provenance graphs that initiate or receive remote connections, assessing their alignment with persistence execution and persistence setup as per our advanced detection rules. This approach enables CPD to identify potential persistence activities through detection rules and to analyze processes engaged in remote connections, assessing their role in the broader scheme of a persistence threat. This dual-layered strategy, combining precise activity tracking with the creation of pseudo-edges, empowers CPD to surpass existing threat detectors.

Despite the advantages, relying solely on pseudo-edges for enhancing persistence detection accuracy presents challenges. One major issue is that even the most comprehensive logging systems may not capture all connections, potentially leading to gaps in the provenance graph and, consequently, false negatives. Our research found that integrating Windows ALPC logs with system audit logs could bridge these gaps. However, this integration significantly increases log storage requirements. Instead of depending on ALPC logs, CPD employs a novel technique of *expert-guided edges*, utilizing insights from process creation and system policy to refine the provenance

graph and reduce log volume by an average of 37%. While pseudo-edges specifically aid in identifying persistence threats, expert-guided edges serve a broader purpose, improving overall efficiency in tracing and log management without being limited to detecting persistence threats.

Another challenge arises from some benign programs' behaviors that cause pseudo-edges to be generated excessively, which can complicate the detection process. To counteract this, CPD incorporates a sophisticated false positive reduction algorithm, which is informed by an in-depth analysis of APT behaviors. This ensures that only genuinely malicious activities are flagged. Our algorithm capitalizes on the crucial understanding that persistence is merely one component of a multi-stage APT kill chain, all of which need to be executed in concert to fulfill the attackers' objectives. By verifying the presence of related kill chain techniques and tactics within close proximity in the provenance graph, CPD significantly enhances its capability to distinguish between benign and malicious actions, improving both the accuracy and reliability of threat detection.

Our system CPD outperforms existing detection systems, as evidenced by evaluations on both public datasets and those derived from strictly implemented MITRE emulation plans [36]. It excels in reducing FPs by 93% on average, effectively detecting true persistence attacks, and producing succinct attack graphs that explain the persistence setup and execution. The capability to pinpoint persistence setup and execution within a provenance graph and present it in context provides security analysts with actionable insights for further investigation, demonstrating CPD's practicality. Mostly, CPD has a response time of under a minute for its entire pipeline.

Unlike previous techniques, CPD produces alarms satisfying all five properties of reliability, explainability, analytical depth, contextuality, and transferability as introduced in [32]. In its first stage, CPD's detection rules avoid easily changeable indicators, such as hard-coded IP addresses and file hashes, which are common in rule-based security systems [21]–[23], ensuring reliable detection. The attack graphs produced in the second stage of CPD are both explainable and contextual, providing an analytical overview of the attack. Finally, the system's customizable weighting factors for indicators and the alert budget introduced in the third stage make CPD highly transferable and adaptable for practical use.

The main contributions of our paper are:

- We present a thorough analysis of cyber persistence attacks and propose CPD, the first detection system specifically targeting persistence threats.
- We introduce **pseudo-dependency edges** to causally relate disjoint persistence phases and improve the detection of these threats.
- We propose the novel concept of **expert-guided edges** to enable efficient provenance tracing of persistence threats.
- We identify critical insights on determining malicious persistence behaviors and propose **an alert triage algorithm** incorporating these insights to reduce false positives.
- We implement and evaluate CPD on diverse datasets, demonstrating better attack detection rates and provenance graph completeness versus state-of-the-art methods.

II. Motivation

A. APT Attack Stages

In a typical APT attack campaign, attackers gain *initial access* to a victim organization mainly through program exploits or stolen credentials. Program vulnerabilities are often patched, and users frequently change passwords, making these methods unreliable for long-term operations. Thus, attackers *establish persistence* using various methods for prolonged, reliable access. Post-persistence, they perform *local discovery* to understand target systems, including security program details. To *evade detection*, they select tools to bypass these security programs or deactivate them after *privilege escalation*. Afterwards, attackers conduct *network discovery* to find and *move laterally* to other vulnerable machines. The hallmark of APT campaigns is not immediate impact but remaining undetected for long periods, aiming to *collect* and *exfiltrate data* or cause significant *impact* at a strategic point. The ability to achieve persistence is critical for attackers' success.

B. Persistence Prevalence

We extracted data from MITRE ATT&CK knowledge base [37], which includes adversary tactics and techniques based on real-world observations. Our statistical analysis on MITRE's database sheds light on the prevalence of persistence (sub-)techniques in the wild. There are 99 distinct (sub)-techniques in persistence tactic. We find out that 94 out of 136 APT groups (69%) leveraged at least one persistence (sub-)technique in the past. We rank both persistence (sub-)techniques based on the number of APT groups that leveraged these techniques in the real world, and APT groups based on the number of persistence (sub-)techniques employed by them. Due to space limit, we only show the top 10 persistence (sub)-techniques in Table I, and top 10 "persistent" APT groups in Table II.

C. Why is Persistence Misunderstood in APT Detection?

Persistence is often seen in the wild, but ironically, despite its emphasis in the term APT, it is misunderstood in academic research. MITRE defines persistence as the ability to maintain access to victim systems across restarts, changed credentials, and other interruptions that could (temporally) cut off access. This can be achieved in two primary ways: either by attackers initiating a remote connection using stolen credentials (T1078.003) or placing a public key in the `SSH authorized_keys` file (T1098.004); or by the attackers initiating a connection from within the victim system, such as by placing a new entry in the Registry run keys or startup folder (T1547.001), creating scheduled tasks or jobs (T1053), or inserting commands in Unix shell configuration files like `.bashrc` (T1546.004). For further details regarding these techniques, we refer the reader to [35].

In our review of all public audit log datasets [38]–[40] for evaluating PIDS, we identified a significant gap in the understanding of persistence. Effective persistence hinges on three key conditions: setting a trigger (like creating a Registry run key), linking code for remote connection to this trigger

TABLE I: Top 10 persistent (sub-)techniques

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

TABLE II: Top 10 persistent APT groups

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

(e.g., placing an executable in the Registry run key), and the successful initiation of a remote connection when the trigger is activated. In practice, the third condition may not always be met, prompting APT actors, such as APT29 to deploy multiple persistence techniques to enhance their odds of maintaining presence in the target environment. Unfortunately, existing datasets often fulfill only the first condition, setting up a trigger, and frequently link an irrelevant value to it, thereby missing the second and third conditions. This oversimplification can be counterproductive, as normal programs often activate these persistence triggers, enabling attackers to blend into routine system activities. Unlike these datasets, each of MITRE's eleven emulation plans [36], based on real-world APT behaviors, incorporates at least two persistence techniques without such oversimplification. Thus, we utilize these emulation plans to evaluate CPD as detailed in Section VI.

III. Threat Model

Our system, like other PIDS [9]–[16], considers firmware, OS, and our logging systems in trusted computing base (TCB). Unlike [9], [13]–[15], we do not presume attacks end before an OS reboot. Our system is unique in linking provenance attack graphs across reboots using *pseudo-edges*. Our logging, as a Windows service or a `Systemd` service on Linux, restarts post-reboot, but an OS reboot may miss system audit events. Notably, attack-related processes starting before our logging service are not logged, often indicating persistence attacks. On Windows, persistence can be achieved through "Create or Modify System Process: Windows Service" (T1543.003), and on Linux through "Create or Modify System Process: Systemd Service" (T1543.002). We can trace back to the root process in logs despite missing initial process creation events, using parent process GUID/ID. Our tools System Monitor [41] on Windows and Auditd [42] on Linux record these IDs.

Technique Coverage. Of those 99 distinct (sub-)techniques in persistence tactic, only half of them were knowingly used by at least one APT group in the past according to MITRE's database. We only created detection rules for about a third of all persistence (sub-)techniques, which all fall into the "significant" half of persistence (sub-)techniques and include all top 10 persistence (sub-)techniques. We have excluded all persistence (sub-)techniques related to macOS, cloud infrastructures, pre-OS boot like Bootkit.

IV. System Design

CPD employs a four-step approach for accurate persistence detection, as illustrated in Figure 2. Initially, CPD processes system event logs to create a *persistence setup table*, recording activities that match persistence setup detection rules, such as new account creations or Registry run key additions. Next, it identifies processes with remote connections in the event stream, performing individual backward tracing to generate sub-graphs for each. These sub-graphs are then checked against our persistence execution detection rules, with matches recorded in a *persistence execution table*. In the third step, CPD aligns entries from persistence execution table with persistence setup table based on TTP labels[1], temporal order, and specific attributes. It creates a *persistence setup atomic graph*, i.e., a minimal sub-graph directly related to persistence, and a *persistence execution atomic graph* when an alignment is found, and then links them with a pseudo-edge. For evaluation purposes (Section VI), steps 2 and 3 are combined as step 2 alone does not yield direct alerts. The final step introduces *pseudo-edge strength* and a false positive reduction algorithm to ensure only significant events are connected.

A. Persistence Threat Detection

Our persistence setup detection rules are created by studying persistence (sub-)techniques described in MITRE ATT&CK Matrix, hundreds of persistence-related threat reports, and red team tools that provide visibility into low-level code related to persistence attacks, like Atomic Red Team [43]. We then improved our persistence setup detection rules by studying and incorporating open-source detection rules from popular rule repositories like Sigma [23] and Elastic [21]. However, as discussed in Section VI-A in details, our rules are different from the ones in those repositories. We will commit our persistence detection rules to these open-source rule repositories for the public's benefit.[2] The persistence setup detection process is mostly a straightforward string match process against information inside a single system log event. But some rules have a few more rule conditions, which require information across several log events. For this, we use "sequenced" query of EQL, a query language specifically designed for threat hunting [44]. Indicative strings include especially file paths, Registry locations, process names and command lines etc. Note that these strings are characteristic to the persistence (sub-)techniques. For instance, to implement

[1]TTP stands for Tactics, Techniques, and Procedures. An exemplary TTP label is T1547.001.

[2]The first author of this article has been submitting accepted commits for non-persistence-related TTP detection rules to those repositories.

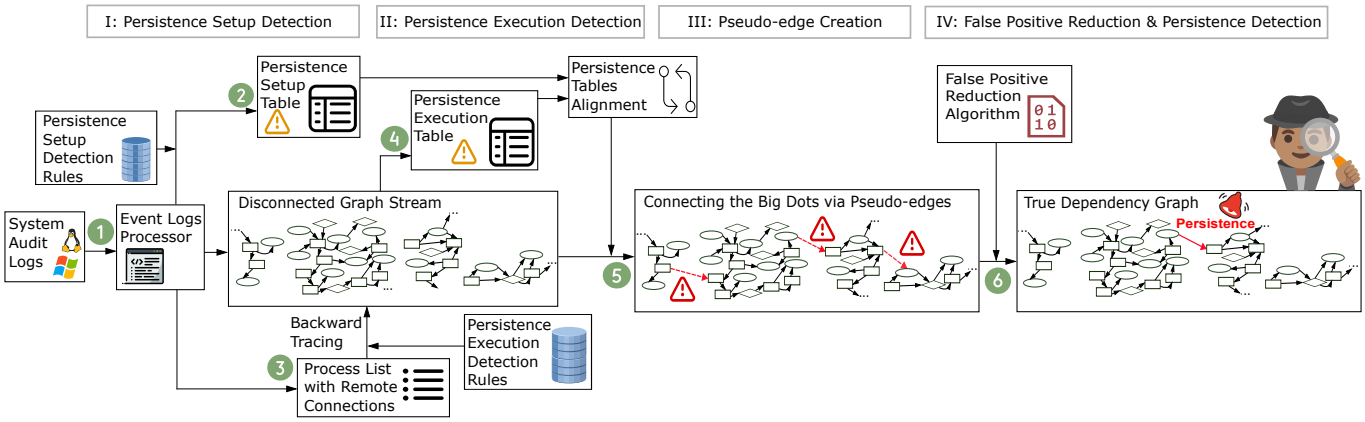

Fig. 2: CPD overview. CPD implements a four-step approach for detecting persistence threats, starting with the creation of a persistence setup table from audit logs that tracks potential setup actions. It then traces processes with remote connections to form sub-graphs, which are evaluated against execution rules and aligned with setup actions to form atomic graphs linked by a pseudo-edge. The process is refined through the introduction of pseudo-edge strength and a false positive reduction algorithm.

the persistence sub-technique T1547.001 (Registry run keys), one of a few known Registry locations *must* be modified.

Simply matching system events against those detection rules generates a huge volume of alerts in practice. To reduce false alerts, CPD first spots every process initiating or accepting remote connection(s) in the event stream, then performs backward tracing on these processes individually. During the backward tracing, CPD inspects whether its provenance graph contains system activities matching our persistence execution detection rules. Like persistence setup detection rules, our persistence execution detection rules contain other indicative strings incorporating file paths, Registry locations, process names etc. These strings are also characteristic to the persistence (sub-)techniques. For example, to "activate" the persistence sub-technique T1547.001 (Registry run keys), one of a few known Registry locations *must* be read by exactly the process `explorer.exe`, and it *must* rely on this process to (directly or indirectly) start the malicious process.

Table III demonstrates how sensitive these detection rules are. Note that we assume the integrity of the operating system including its native built-in system programs. We stress that, at its stage 1, CPD sacrifices specificity for sensitivity in its detection results, in order to not miss a single potential persistence attack. In other words, CPD will not have false negatives at this stage, but at the cost of having many false positives. The optimization techniques introduced in CPD and discussed in Section VI-A only improve its stage 1's specificity, and do not impact the sensitivity. Besides, we argue that the evasion techniques for SIEM (Security Information and Event Management) rules introduced in [45] have only limited impact on our detection rules. Because, as discussed in Table III, we mostly do not rely on recorded command lines or code executed by attackers. Rather, we use indicative file paths, Registry locations and system process names. These strings are immutable under the assumption of OS integrity.

For every detected potential malicious process (with remote connections) from above, CPD checks if it has corresponding entries in the persistence setup table, based on the TTP labels, some TTP-specific attributes, and happens-before relationship (Algorithm 1 Lines 4-9). If an alignment is found, a persistence execution atomic graph is created (Algorithm 1 Line 10), which includes only information related to persistence execution. Likewise, a persistence setup atomic graph containing only critical attack information is also generated. Then CPD creates a pseudo-edge to connect the persistence setup atomic graph and the persistence execution atomic graph (Algorithm 1 Line 12), resulting into a succinct and insightful persistence attack graph, as shown in Figure 3.

Algorithm 1: PSEUDO-EDGE CREATION

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

15 Function GetPersistenceSetup(E)
16     foreach Rule ∈ L_PersistenceSetupRule do
17         forall Condition ∈ Rule do
18             satisfied ← CheckCondition(Condition, E)
19             if satisfied then
20                 L_<Eα,Lα,Tα> ← L_<Eα,Lα,Tα> ∪ (E,L,T)
21     return L_<Eα,Lα,Tα>

22 Function GetPersistenceExecution(Pγ)
23     L_<Eκ> ← TraversalBackward(Pγ, E)
24     foreach Eκ ∈ L_<Eκ> do
25         foreach Rule ∈ L_PersistenceExecutionRule do
26             forall Condition ∈ Rule do
27                 satisfied ← CheckCondition(Condition, Eκ)
28                 if satisfied then
29                     L_<Eα,Lα,Tα> ← L_<Eα,Lα,Tα> ∪
                         (Eκ,Lκ,Tκ)
30     return L_<Eγ,Lγ,Tγ>
```

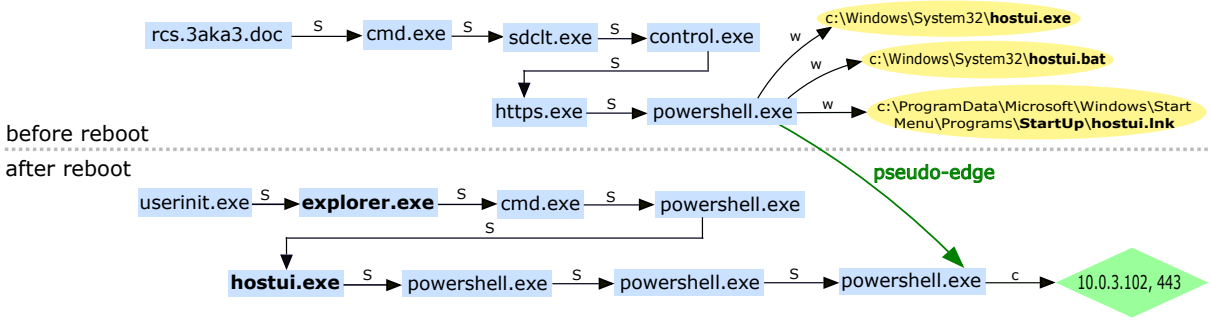

Fig. 3: A persistence attack graph automatically generated by CPD on the EP-APT29-1 dataset. It uses rectangles for processes, ovals for files / Registry keys, and diamonds for network sockets. Annotations include S=Start, W=Write, C=Connect. The graph successfully pinpoints T1547.001 (Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder). The upper section reveals persistence setup: a malicious Microsoft Word-like program (`.doc`) starts, resulting in a Powershell instance and a shortcut creation in the Windows startup folder. This shortcut leads to another dropped malicious program, `hostui.exe`. The lower section, post-reboot, shows persistence execution: `explorer.exe` auto-executes startup folder shortcuts, triggering malicious Powershell code and connecting to the attacker. Indicative strings are bolded for clarity. CPD forms a pseudo-edge linking the process initiating persistence setup with the one managing the remote connection, i.e., the c2 agent.

TABLE III: Detection rule sensitivity for the top 10 persistent techniques. TPR = True Positive Rate, ✓= Yes, ✓̸= Almost.

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

B. Expert-guided Edges

In our experiments, we found limitations in linking system entities solely based on Windows' Process Monitor / System Monitor logs. Specifically, during T1543.003 (Create or Modify System Process: Windows Service) persistence setup, attackers often use `sc.exe` to create a malicious Windows service. This results in a new Registry key under `HKLM\SYSTEM\CurrentControlSet\Services`, the basis for our detection rule. This approach, focusing on an immutable Registry location, is more reliable than relying on command lines, which are easily bypassed, as recent research shows [45].

In the logs, the corresponding Registry key modifications appear to be done by `services.exe`, not `sc.exe`, with no apparent link between the two. Further research and consultation with the Windows Developer Reference [46] revealed that the communication between these processes occurs through ALPC, a Windows inter-process communication (IPC) method not typically logged by standard frameworks. Additional logging via Windows ETW "NT Kernel Logger" confirmed the link but resulted in excessively large datasets due to ALPC's widespread use.

To address this, we introduce *expert-guided edges* in CPD. These edges are formed by applying specialized parsing rules

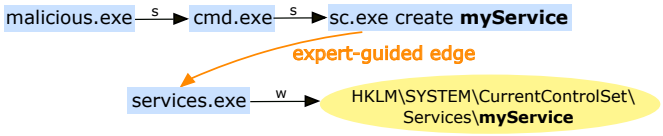

Fig. 4: An expert-guided edge is created during reconstruction of a T1543.003 persistence setup attack graph. An attacker-controlled malicious process leverages LOLBins to create a malicious service for persistence. The indicative Registry key is however modified by a Windows system process, to which no link from the malicious process can be built using logs from standard logging frameworks.

Algorithm 2: EXPERT-GUIDED EDGE CREATION

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

1 **foreach** $\mathcal{E}_\kappa \in \mathcal{E}$ **do**
  /* Get the process of current event */
2 $\mathcal{P}_\kappa \leftarrow$ GETSUBJECT($\mathcal{E}_\kappa$)
3 **if** $\mathcal{P}_\kappa \in L_{<\mathcal{P}>}$ **then**
    /* Add dependency path to the standard routine nodes */
4 $P \leftarrow$ ADDPATHTOROUTINENODES($\mathcal{P}_\kappa$)
5 $L_{<P>} \leftarrow L_{<P>} \cup P$
6 **return** L_P

during log processing for provenance graph generation. This method embeds expert knowledge about process creation routines and operating system policies into the backward and forward tracing process. For example, we can link `sc.exe` and `services.exe` if `services.exe` modifies a Registry key under the specified location right after `sc.exe` is executed with the same service name, as illustrated in Figure 4. This approach offers three benefits: faster search process, reduced dependency explosion, and bridging gaps otherwise impossible to close.

Similarly, we observe missing links on Linux using Auditd logs even though we are monitoring a very extensive list of syscalls. Specifically, during T1543.002 (Systemd Service) persistence setup, an indicative file `/etc/systemd/system/*.service` is created. However, during persistence execution, we cannot observe the same file being accessed, but rather a closely related in-memory file `/sys/fs/cgroup/system.slice/*.service/*` with the same service file name. An edge cannot be built between the corresponding sub-graphs if using the traditional "write and read on the same file node" principle. Hence we use an expert-guided edge to resolve this issue. Algorithm 2 describes the creation of expert-guided edges, and is applied in Line 23 of Algorithm 1. Note that we create expert-guided edges under the assumption that the integrity of the OS itself is not compromised. We show the log storage reduction rate by introducing expert-guided edges in Section VI-B.

C. False Positive Reduction

Creating a pseudo-edge that accurately indicates a persistence attack is challenging. This is mainly due to the fact that many benign programs use Registry run keys (T1547.001), Windows services (T1543.003), scheduled tasks (T1053.005) etc. for some program-specific routines involving remote connections, leading to an inadequate amount of false-positive persistence attack graphs. For instance, some benign programs create a Windows service to check and download updates periodically, like Adobe Acrobat's Update Service (`armsvc.exe`), mimicking persistence behaviors and triggering false-positive pseudo-edges in CPD. While the Windows service creation itself mimics persistence setup, the resulted update downloads (involving remote connections) later on mimic persistence execution. Similarly, programs such as Google Chrome and Microsoft OneDrive use Registry run keys for updates, frequently leading to false alarms. Figure 5 shows a typical false-positive persistence attack graph generated by CPD after its stage 2.

To address this, we developed a false positive reduction algorithm using contextual indicators to differentiate between benign and malicious activities. This algorithm introduces *pseudo-edge strength* and ranks pseudo-edges based on a calculated threat score. We categorize pseudo-edges as either causality-based or correlation-based for improved detection accuracy, due to the difference in the nature of various persistence techniques. Correlation-based pseudo-edges, related to login account techniques like T1098, T1136, and T1078, are less reliable due to the uncertainty of user identity behind consecutive logins. These receive a 'penalty' weight (less than 1) in the anomaly score assignment (Equation 3). For precise detection, we utilize context not only from the cyber-kill-chain tactic and technique levels, but also from the program execution level.

1) Causality-based pseudo-edges: We formulate the following indicators based on studying APT behaviors in real-world attacks.

- **Degree of indirection** in both persistence setup and persistence execution. As observed in APT29's real-world behaviors, multiple indirection is implemented to start the command and control (c2) agent program. That is, when the malicious shortcut file in a Windows startup folder is read, the `explorer.exe` process runs a normal-looking batch file linked to the shortcut file (indirection 1). The resulted `cmd.exe` process starts a `powershell.exe` process as instructed in the batch file (indirection 2), which in turn starts another normal-looking process (indirection 3). This process then starts another `powershell.exe` process (indirection 4), which again starts another `powershell.exe` process (indirection 5) before contacting the c2 server. This obviously deviates from normal programs' use of Windows startup folders.
- **Credential access tactic**, as observed in APT actors' past behaviors, e.g., APT29 [47], Sandworm [3], Wizard Spider [48], is almost always executed as an attempt to obtain the "low-hanging fruits" persistence.
- **Persistence techniques are often executed together**, as they together will likely contribute to more reliable persistence.
- APT actors tend to execute persistence techniques **right after initial access or lateral movement**. For example, Wizard Spider achieved persistence right after initial compromise on the first victim machine, and then persistence on the second victim machine right after lateral movement.
- APT actors tend to execute some **discovery techniques**

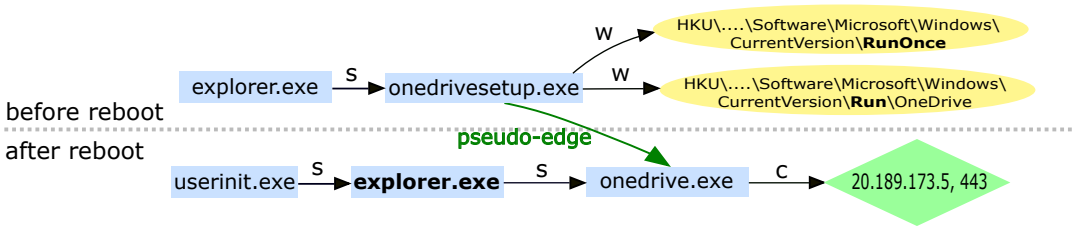

Fig. 5: A false-positive persistence attack graph automatically generated by CPD on the EP-APT29-1 dataset. This graph wrongly classifies an instance of T1547.001. It turns out to be a benign program, i.e., Microsoft OneDrive, leveraging Registry run keys for updates. It in fact connects back to an IP address belonging to Microsoft Corporation.

before persistence techniques. However, we find that this indicator tends to be less reliable and more "noisy" than other indicators. To counter this, we assign the smallest weighting factor to this indicator during anomaly score calculation in Equation 3.

2) Correlation-based pseudo-edges: We further classify correlation-based pseudo-edges into two types.

Type 1 - persistent initial re-access We identified the following indicators for this type of persistence.

- Observation of **credential access tactic** is also an indicator for correlation-based pseudo-edges. For instance, OS credential dumping (T1003) is often performed on victim computers, and the stolen credentials are used for the remote re-connection. Besides, the **usage intensity**, i.e., occurrence of the same technique, and **extensiveness**, i.e., variation of attempted credential access techniques, can be a weighting factor. For example, during one of Sandworm's engagements, they uploaded an executable to its target machine for dumping web credentials, and another executable for key-logging a valid user RDP session to obtain domain credentials.
- The accessing computer does not have a **legit FQDN** or **computer account** in the domain. This is more likely a malicious persistent initial re-access, as attackers typically do not physically own a domain joined computer.
- **Local account creation** after lateral movement, e.g., in a WinRM [49] session.
- **Installing, activating and enabling standard remote access tools** like VNC and RDP server. For instance, Carbank [50] installed a VNC server on its victim machine for persistence after capturing credentials, and opened the corresponding port on firewall.

Type 2 - persistent lateral movement Type 2 correlation-based pseudo-edges are a special kind of pseudo-edges, which represent an intersection between persistence tactic and lateral movement tactic. The following indicators are extracted from analyzing real-world APT attacks.

- **Remote system discovery** (T1018) is performed on one domain-joined computer before a remote connection is initiated from this computer to another computer in the network. For instance, APT29 used LDAP queries to enumerate other hosts in the domain before creating a remote Powershell session to a secondary victim.
- **Ingress tool transfer** (T1105) is performed from one domain-joined computer to another computer before the remote connection.
- **Credential access tactic** is performed on one domain-joined computer before a remote connection. For instance, Sandworm has stolen SSH keys on the first compromised computer and then use these credentials to move laterally to a second computer.
- **Local account creation** after or during lateral movement.
- **Installing, activating and enabling standard remote access tools** like VNC and RDP server.

Taking into account above indicators, CPD calculates an anomaly/threat score, and uses this score to quantify pseudo-edge strength and rank pseudo-edges. By doing so, we ensure that the most likely malicious pseudo-edges are always investigated first by a security analyst. During the final stage of CPD, as formulated in Algorithm 3, a pseudo-edge is first classified into one of the three categories above (Line 4). Then the corresponding persistence attack graph is automatically 1) analyzed on to extract features related to indicators from above, e.g., the degree of indirection during both persistence setup and persistence execution, and 2) further explored on to include more contextual information and search for the existence of other indicators from above, i.e., related attack steps in a cyber-kill-chain, e.g., whether credential access is observed in its dependency graph. The dependency graph expands on the persistence attack graph, and is therefore more verbose. That is, our false positive reduction algorithm takes as input all indicators found in the succinct persistence attack graph as well as in its more verbose dependency graph. In the following, we explain how these indicators are adopted for anomaly score assignment in three equations.

First, we calculate the anomaly score of an indicator observable from the persistence attack graph as follows (Line 8 in Algorithm 3):

$$AS_{ind-PAG} = N_s^2 \times N_e^2 \tag{1}$$

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

Second, the anomaly score of an indicator observable from the dependency graph is calculated as follows (Line 18 in Algorithm 3):

$$AS_{ind-DG} = \begin{cases} \max_i(\frac{D_e}{D_s} \times Freq(teq_i) \times Var(tac)) & t_{teq} \leq t_e \\ \max_i(\frac{D_e}{D_e} \times Freq(teq_i) \times Var(tac)) & t_{teq} > t_e \end{cases} \tag{2}$$

where D_s denotes the distance between an indicative attack step, e.g., OS credentials dumping, and the persistence setup

Algorithm 3: FALSE POSITIVE REDUCTION

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

1 **foreach** $< PE(\gamma,\alpha), PAG(\gamma,\alpha) > \in L_{<PE,PAG>}$ **do**
2 $AS_{PE(\gamma,\alpha)} \leftarrow 0$
3 $L_{<AS_{PE}>} \leftarrow 0$
/* Classify pseudo-edge */
4 $PE'(\gamma,\alpha) \leftarrow$ GETCATEGORY($PE(\gamma,\alpha)$)
/* Select indicators based on pseudo-edge type */
5 $L'_{<ind-PAG>} \leftarrow$ GETINDICATORS($PE'(\gamma,\alpha)$, $L_{<ind-PAG>}$)
6 $L'_{<ind-DG>} \leftarrow$ GETINDICATORS($PE'(\gamma,\alpha)$, $L_{<ind-DG>}$)
7 **foreach** $indicator \in L'_{<ind-PAG>}$ **do**
8 $AS_{indicator} \leftarrow$ CALCULATESCORE1($indicator$, $PAG(\gamma,\alpha)$)
9 $L_{<AS_{PE}>} \leftarrow L_{<AS_{PE}>} \cup AS_{indicator}$
10 $(L_{<\mathcal{E}\delta>}, DG(\delta)) \leftarrow$ TRAVERSALBACKWARD($PAG(\gamma,\alpha)$, $\mathcal{E}$)
11 $(L_{<\mathcal{E}\eta>}, DG(\eta)) \leftarrow$ TRAVERSALFORWARD($PAG(\gamma,\alpha)$, $\mathcal{E}$)
12 $DG(\kappa) \leftarrow$ MERGEGRAPH($DG(\delta)$, $DG(\eta)$)
13 $L_{<\mathcal{E}\kappa>} \leftarrow L_{<\mathcal{E}\delta>} \cup L_{<\mathcal{E}\eta>}$
14 **foreach** $\mathcal{E}_\kappa \in L_{<\mathcal{E}\kappa>}$ **do**
15 **foreach** $indicator \in L'_{<ind-DG>}$ **do**
16 $satisfied \leftarrow$ CHECKINDICATOR($indicator$, $\mathcal{E}_\kappa$)
17 **if** $satisfied$ **then**
18 $AS_{indicator} \leftarrow$ CALCULATESCORE2($indicator$, $DG(\kappa)$)
19 $L_{<AS_{PE}>} \leftarrow L_{<AS_{PE}>} \cup AS_{indicator}$
20 $AS_{PE(\gamma,\alpha)} \leftarrow$ SUMSCORE($L_{<AS_{PE}>}$)
21 $L_{<PE,AS>} \leftarrow L_{<PE,AS>} \cup AS_{PE(\gamma,\alpha)}$
22 $L_{<PE,AS>} \leftarrow$ SORTBYSCORE($L_{<PE,AS>}$)
23 $L_{<PE,AS>} \leftarrow$ REMOVEBYBUDGET($L_{<PE,AS>}$, $\mathcal{N}$)
24 **return** $L_{<PE,AS>}$

step. The distance is measured as the number of hops between the corresponding two processes. Similarly, D_e denotes the distance between the persistence execution step and an indicative attack step, e.g., remote system discovery. D_c is a predefined cut-off value representing the maximal number of hops considered as having positive impact on the anomaly score. By doing so, it penalizes an indicative attack step too far away from the persistence setup or execution step, in which the weighting factor in Equation 2 is less than 1, when D_s or D_e is greater than D_c. $Freq(teq)$ represents the occurrence of the same attack technique being executed as repeated attempt, whereas $Var(tac)$ represents usage extensiveness of the same tactic, i.e., number of different techniques from the same tactic being applied. The rationale behind this is that attackers often try a variety of techniques from the same tactic together to maximize the chance of achieving their objectives. t_{teq} is the time when an attack step is conducted, and t_e is the time when persistence execution is performed. If multiple attack (sub-)techniques are observed for the same indicator, we only consider the one with the maximal anomaly score.

In the end, we obtain the final anomaly score from the Equation 3 (Line 20 in Algorithm 3).

$$AS_{PE} = \prod_{i=1}^{n} (AS_i)^{w_i} \tag{3}$$

where n denotes the number of found indicators for a given pseudo-edge, AS_i denotes the anomaly score of an indicator obtained from Equation 1 or Equation 2 for this pseudo-edge, and w_i is a weighting factor. Afterwards, the pseudo-edge and anomaly score pair is added to a list (Line 21 in Algorithm 3), which is sorted by anomaly score at the end (Line 22 in Algorithm 3). Pseudo-edges ranked lower than the $\mathcal{N}$-th pseudo-edge are considered as false-positive pseudo-edges, and therefore removed from the list (Line 23 in Algorithm 3). Algorithm 3 returns the final list of pseudo-edge and anomaly score pairs.

V. IMPLEMENTATION

For our experiments, we implemented the prototype of CPD in Python ($\sim$ 6K lines of code), and deployed it on a 64bit Ubuntu 23.04 OS with 512 GB of RAM and a 64-core AMD processor. This machine hosts a dozen virtual machines used by several researchers for conducting separated scientific experiments at the same time. Our implementation interfaces Elasticsearch [51] via EQL, a query language specifically designed for security use cases. Elasticsearch provides scalable and near real-time search for log data investigation. Besides, we use Python NetworkX [52] for generating provenance graphs on demand, and PyVis [53] for graph visualization.

For Linux log collection, we use Auditd [42]. On Windows, our primary tool is System Monitor (Sysmon) [41], which, unlike Windows ETW, generates and records a process GUID for each process, reducing false dependencies during post-processing. However, Sysmon lacks file/Registry read collection, so we use Windows Security Audit logs for recording all file and Registry operation. We also collect ALPC logs with the "NT Kernel Logger" [54] ETW session.

VI. EVALUATION

In this section, we evaluate the efficacy and effectiveness of CPD as a persistence detection system. In particular, we investigate the following research questions (RQs):

- ***RQ1*** How does CPD compare in soundness of persistence detection, false positive reduction, and accuracy to open-source SIEM detection rules, commercial EDR systems and state-of-the-art PIDS? (VI-A)
- ***RQ2*** How high is the log reduction rate by introducing expert-guided edges in CPD? (VI-B)
- ***RQ3*** What is the runtime overhead of CPD? (VI-C)
- ***RQ4*** How precise are persistence attack graphs generated by CPD? (VI-D)

Public Datasets. Public datasets often lack persistence traces. From DARPA datasets, we chose the E5 dataset [39] and OpTC dataset [38], but omitted the E3 [55] due to its lack of persistence attacks. DARPA E5 dataset features emulated APT attacks. Only the E5 Fivedirections subset, focused on Windows, contains two instances of persistence attacks. We evaluated only this subset with CPD. The DARPA OpTC

TABLE IV: Overview of the evaluation datasets

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

dataset contains logs from 500 Windows machines. It includes three persistence instances, but only one meets all criteria from Section II. The other two failed the third condition as the attack ended prematurely. We tested CPD on a subset of 50 machines, including the 3 with persistence and 47 random ones. We also included ATLASv2 [56] for its CBC detection results on persistence. ATLASv2 dataset offers more background activities and extensive logging than ATLAS [40], such as through Sysmon and VMware CBC [57]. While lacking actual persistence attacks, it includes CBC's persistence alerts. We used these for comparison with CPD. CPD's detection results on these datasets are verified against the provided ground truth.

MITRE Attack Emulation. MITRE's eleven full emulation plans [36], based on real APT behaviors, each include at least two persistence techniques. These plans are used in MITRE Engenuity ATT&CK® Evaluations [58] to assess commercial EDR systems [59]. However, MITRE has not published any corresponding datasets. We precisely implemented two relevant emulation plans, focusing on top 10 most "persistent" APT groups from Section II-B, and then evaluated CPD on these datasets, valuable for PIDS research. MITRE's emulation plans target enterprise networks with more sophisticated, cross-machine attacks than most public datasets, offering greater authenticity. The APT29 plan has two distinct scenarios, while Sandworm's are identical. We created three datasets from emulating APT29 scenario 1, APT29 scenario 2, and Sandworm scenario 1, named EP-APT29-1, EP-APT29-2, and EP-Sandworm-1, respectively. Table IV gives an overview of our datasets.

A. *Effectiveness of CPD*

CPD vs. SIEM detection rules. After finalizing our persistence detection rule set, we compared it with popular open-source SIEM detection rules from Elastic [21], Sigma [23] and Google Chronicle [22]. We extracted all persistence-related detection rules from these repositories, converted them into EQL queries and ran them alongside CPD on datasets containing persistence attacks. The results in Table V reveal that open-source SIEM rules generated more false positives and missed true attacks, and CPD significantly outperformed these rules. Further comparison between CPD's stages 2 and 3 is presented in Figure 6. This figure displays the cumulative distribution of threat scores for benign and attack pseudo-edges. The stage 3 results in Table V are based on the lowest true attack threat score threshold. CPD's final stage drastically reduced false positives—from thousands in open-source rules to just dozens per dataset. Additionally, the false positive reduction algorithm in stage 3 of CPD achieved an average reduction rate of 83%, significantly enhancing accuracy.

CPD's great improvement over SIEM detection rules mainly result from our two key insights, implemented on its stage 2 and 3 respectively. Our first powerful insight is that all persistence attacks require a setup stage that (mis)uses a "sensitive" system functionality and a subsequent execution stage linked with a remote connection. Both SIEM rules and CPD's stage 1 rules raise alerts when persistence-related system functionality is used, but not necessarily misused. Unlike SIEM rules, which don't associate log events generated at different times, CPD's stage 2 checks if there is a remote connection that can be traced back to a setup alert, leading to removal of alerts on system activities associated with benign usage of system functionality. Our second practical insight is that persistence is only one of multiple stages in a cyber-kill-chain that must be executed together to achieve attackers' goals. This key insight is discussed in details in our false positive reduction algorithm in Section IV-C. Both key insights fully leverage the power of provenance analytics, i.e., context provided by system activities across a large timespan.

All parameters in our false positive reduction algorithms, e.g., weighting factors, are unchanged for each dataset. Like previous works [13]–[15], our goal is not to exclude all FPs, but rather to prioritize most potential attacks for investigation using threat score ranking. In other words, CPD aims to maximize the likelihood of detecting persistence attacks based on limited human resources. Based on the alert budget security analysts have, they can flexibly configure this parameter.

We further investigate on why these open-source detection rules have undesired outcomes, in particular, 1) why there are many false negatives when applying Elastic's rules; 2) why Sigma's rules create exceptionally high number of false positives. To answer the first question, we carefully inspect Elastic's persistence detection rules. We find out that Elastic's rules not only overly allowlist programs, but also tend to be too specific, containing many hard-coded strings as conditions. Both cases make them very vulnerable to evasion attacks and have many false negatives in practice. For instance, this Elastic rule for T1547.001 [60] allowlists all programs not only under `C:\Windows\System32\` but also under `C:\Program Files\`, essentially excluding the majority of programs on Windows. As discussed above, DARPA OpTC has three persistence instances, including two "unsuccessful" persistence instances only due to incomplete attack scenarios. We find that, due to the high-degree of specificity and overly allowlisting, Elastic's rules missed all these three persistence instances. On the contrary, our system CPD detected all three persistence instances in stage 1, and removed the two unsuccessful persistence instances in stage 2, presenting only the true persistence as its final output.

To answer the second question, we resort to manually inspecting Sigma's persistence detection rules. We find that Sigma's rules are more diverse, with some being overly specific and others being too general. This Sigma rule sample

TABLE V: Comparison of CPD and open-source SIEM rules.

Datasets	CPD						Elastic		Chronicle		Sigma	
	Stage 1		Stage 2		Stage 3							
	FP	FN	FP	FN	FP	FN	FP	FN	FP	FN	FP	FN
DARPA-E5	21911	0/2*	117	0/2	7	0/2	5371	1/2	-	-	57869	0/2
DARPA-OpTC	63460	0/1	35	0/1	4	0/1	15111	1/1	47584	0/1	11567	0/1
EP-APT29-1	4489	0/2	82	0/2	23	0/2	260	2/2	1680	1/2	15158	0/2
EP-APT29-2	3256	0/3	62	0/3	14	0/3	351	2/3	1724	1/3	13305	0/3
EP-Sandworm-1	3881	0/4	48	0/4	8	0/4	527	1/4	1209	0/4	63760	0/4

* The number before / represents false negatives, and the number after / represents true positives in the corresponding dataset.

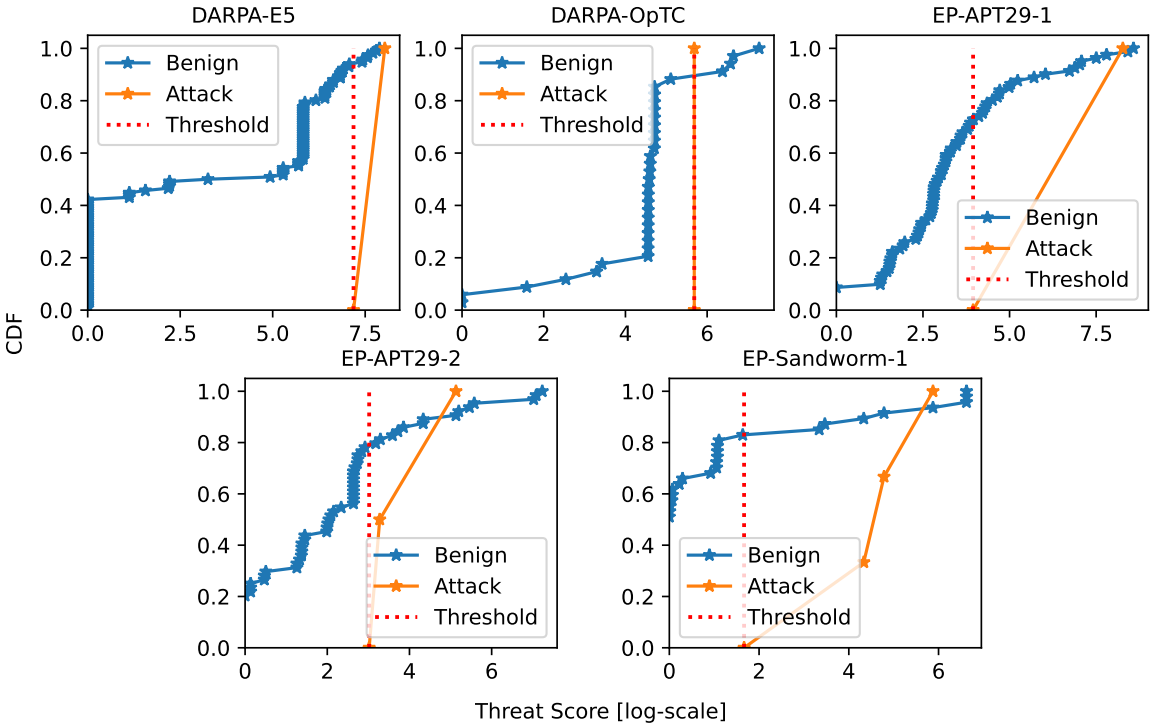

Fig. 6: CDF of threat score for false and true alerts

for T1574.009 [61] uses only a program file path as condition, leading to excessive amount of alerts. We also find that the high number of alerts is partly due to the fact that the Sigma rule repository contains many similar rules created by different contributors for the same attack (sub-)techniques. We argue that the repository maintainers should more properly organize the rules and remove seemingly redundant rules.

In comparison to Sigma, Chronicle's rules cause less alerts, but have a few false negatives. Besides, we also find out that Chronicle has overly simple rules causing too many alerts such as this rule for T1053.005 [62]. At last, Chronicle's rules overly use process names/paths as conditions. However, in the DARPA E5 dataset, process name is absent in log events caused by processes that were created before the logging framework started. Lacking this information will result in wrong results. Hence we do not evaluate Chronicle's rules on the DARPA E5 dataset. It is worth mentioning that this study focuses on detecting persistence, which is often overlooked or even badly understood, as explained in Section II. It is not surprising that those popular detection rule repositories have less sound or complete persistence detection rule sets.

CPD vs. CBC EDR. We then compare our system CPD with the commercial VMware CBC EDR [57] on ATLASv2 dataset. For fair comparison, we did not run CPD's stage 3 on this dataset, because it does not include a real persistence attack. Table VI shows the detection results of CPD and CBC EDR. Like above, without running its stage 3, CPD already outperforms the CBC EDR by reducing the false positive rate by 80%. Besides, we find that CBC's IOC (indicator of compromise) hits reveal that they blindly allowlist programs as well, like SIEM detection rules discussed above. This could easily result in false negatives.

We stress that CPD's stage 1 also eliminates excessive false alarms caused by critical system programs modifying files and Registry at indicative locations. But CPD first checks if those allowlisted programs are potentially compromised by examining whether their executable files are modified. To further optimize the detection results at the stage 1 of CPD, it does

TABLE VI: Comparison of CPD and CBC EDR

Dataset	CPD						CBC	
	Stage 1		Stage 2		Stage 3			
	FP	FN	FP	FN	FP	FN	FP	FN
ATLASv2	1602	0	11	0	-	-	56	0

not create unnecessary alerts for DLL files dropped on disk that get deleted afterwards. Otherwise it would produce lots of security alerts related to several persistence (sub-)techniques, e.g., T1574.001 and T1574.002. This is due to a common Windows program behavior, in which a program drops some DLL files to a (temporary) file folder after being started, then it starts some new instances (as child processes) that load those DLL files. The DLL files get deleted when those child processes terminate. However, unlike Linux, Windows by default never deletes temporary files, and leaves it to the programs for the clean-up. That is, temporary files persist reboots if not deleted by their creator. Hence, if a DLL file is dropped and not deleted afterwards, CPD generates a persistence setup alert for it in its stage 1.

CPD vs. prior PIDS. Most state-of-the-art heuristics-based PIDS [13]–[15] are evaluated on either proprietary datasets or datasets without persistence attacks. Hence it is not possible for us to make direct comparison with them. However, they would, by construction, fail at detecting persistence. Because a forward or backward tracing will not reach an event of interest in the next phase, if attackers break down the entire cyber-kill-chain into multiple phases like in Figure 1. In fact, persistence attacks can be used to totally evade them.

Therefore, we envisioned a comparison with three most recent state-of-the-art learning/anomaly-based PIDS KAIROS [17], FLASH [18] and MAGIC [27], which all show superior performance over other learning-based PIDS like [9], [19], [20], [25], [26]. However, none of these works outline detection results regarding to persistence attacks on each dataset. Each attack graph shown in the original papers was created from a subset of DARPA E3, DARPA E5, or DARPA OpTC dataset. None of those chosen subsets contains a true persistence attack. This is little surprising, as we already discussed above that public datasets often lack persistence traces. True persistence attacks only exist in DARPA E5 Fivedirections subset and DARPA OpTC day 2 subset.

MAGIC is evaluated on DARPA E3 dataset, but not on more recent DARPA E5, OpTC or any datasets containing true persistence attacks. Both KAIROS and FLASH are evaluated on OpTC dataset. Although KAIROS is also evaluated on several DARPA E5 subsets, the Fivedirections subset is not included. KAIROS does not provide information needed to run on DARPA E5 Fivedirections. Both KAIROS and FLASH are a complicated system with many hyper-parameters and model configurations, it is challenging to ensure fair extension of their evaluation to E5 Fivedirections. Hence we took the pre-trained model weights for OpTC as provided by the authors of KAIROS and FLASH, and ran them only on the OpTC dataset and only on system events from the host containing a true persistence attack, respectively.

TABLE VII: Comparison of CPD, KAIROS and FLASH on DARPA OpTC (Host 0501). ✓= Detected, ✗= Not Detected

	Persistence Setup	Persistence Execution	Run Time (minute)*	Mean Memory Consumption (GB)
CPD	✓	✓	2	2.6
KAIROS	✓	✗	630	10.2
FLASH	✓	✗	312	9.1

* From data processing to detection result.

TABLE VIII: Log reduction rate of expert-guided edges

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

Both KAIROS and FLASH perform attack detection at a finer granularity than previous PIDS like UNICORN. KAIROS takes system events as input, splits the entire time line into many time windows, and classifies each time window as benign or malicious, whereas FLASH can classify each node in the provenance graph as benign or malicious. KAIROS's detection result on OpTC shows that it has correctly classified the time window, in which persistence setup was conducted, as malicious, but wrongly classified the time window, in which the corresponding persistence execution happened, as benign. Similarly, FLASH produces a set of malicious nodes, which include the nodes related to persistence setup, but not the nodes responsible for persistence execution. As shown in Table VII, CPD, as a dedicated persistence detection system, requires 4× less memory resource than KAIROS and FLASH, while being more accurate and 315× faster than KAIROS, 156× faster than FLASH. Note that CPD interfaces with Elasticsearch, a scalable and near real-time search engine, for rule matching, and performs provenance analytics only on system events related to persistence setup and execution.

B. Log Reduction via Expert-Guided Edges

As discussed in Section IV, with system logs generated by most popular logging frameworks we observe missing links due to the absence of IPC log events. However, by introducing expert-guided edges, we can cut the dependence on IPC events while still capable of detecting relevant attack steps. During implementation of MITRE full emulation plans, we collect ALPC log events using the "NT Kernel Logger" [54] ETW trace session. Table VIII shows the log reduction rate of CPD by employing expert-guided edges instead of relying on ALPC logs on each dataset.

C. Response Time & Runtime Overhead

We divide the response time of CPD into three parts. Figure 7 presents the cumulative distribution function of response time of CPD in its three stages, respectively. The stage 1 response time is measured per detection rule matching. Figure 7 (a) shows that it takes less than half a second to find alert events in an entire dataset for more than 95% of detection

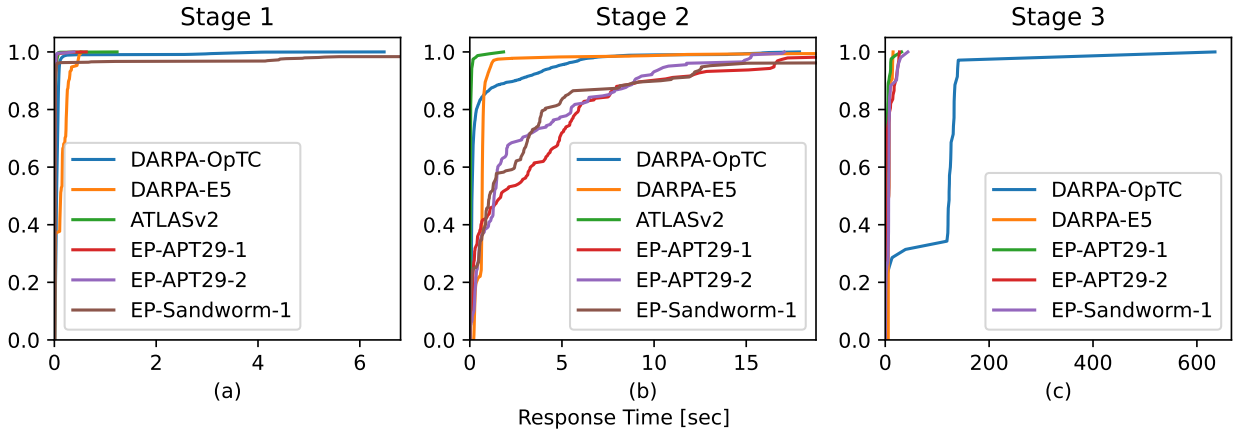

Fig. 7: CDF of response time of CPD

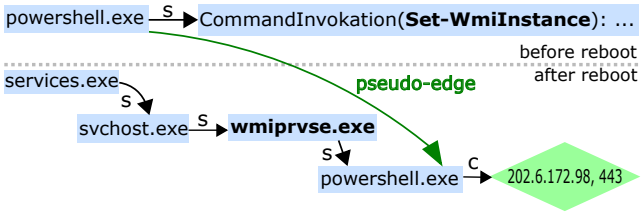

Fig. 8: WMI persistence attack graph automatically generated by CPD on DARPA OpTC dataset.

TABLE IX: Memory utilization (MB) of running CPD

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

rules. We measure the stage 2 response time as the time to perform backward tracing on an indicative process with remote connection(s) while matching against persistence execution rules, persistence tables alignment and pseudo-edge creation. As shown in Figure 7 (b), it takes less than 11 seconds to create a pseudo-edge for over 90% of all indicative processes. The stage 3 response time is the time for performing backward and forward tracing on each alert produced from stage 2 while checking for contextual indicators, and calculating the threat score. Figure 7 (c) shows that it takes less than 140 seconds to inspect each alert for over 95% of all alerts in DARPA-OpTC dataset, and only less than 42 seconds for all alerts in other datasets. Memory overhead caused by running CPD on each dataset is presented in Table IX. The results are measured by using `mprof` [63], which samples memory consumption every 100 ms.

D. Reconstructed Persistence Attack Graphs

An example of reconstructed persistence attack graph from the public dataset OpTC is showed in the Figure 8. This figure depicts a true positive of T1546.003 (Event Triggered Execution: WMI Event Subscription). Like Figure 3 and Figure 5 presented in Section IV, the upper part of the figure shows the persistence setup graph, whereas the lower part shows the persistence execution graph. Both parts are connected via a pseudo-edge. This succinct attack graph exhibits that the attacker created a WMI instance during persistence setup. During persistence execution, the Powershell code for connecting back to the attacker is executed through the Windows system process `wmiprvse.exe`, when a specified event is triggered. We stress that the preciseness of this succinct attack graph containing the most critical information related to persistence can help even inexperienced security analysts achieve a speedy full attack investigation.

VII. Limitations & Discussion

A. Evasion (Mimicry) Attacks

As described in Section III, CPD is not designed to detect all persistence (sub-)techniques. Although CPD is not tested to detect macOS-based persistence (sub-)techniques, we believe the principle is transferable. However, detecting cloud infrastructures-related persistence (sub-)techniques may require a different strategy with cloud infrastructures-specific conditions considered. Like all other provenance-based detection systems, which assume the OS integrity, CPD is not able to detect pre-OS boot persistence attacks like Bootkit. However, none of these unaddressed persistence (sub-)techniques fall into the most misused top 10 persistence (sub-)techniques. That is, we believe this will have a limited impact on the practicality of CPD. Further, CPD is robust against evasion techniques proposed in [28], [29], as they specifically target anomaly-based PIDS that are based on path-based embedding or graph-based embedding. Heuristic-based systems like ours are more difficult to evade, as adding additional "camouflaging" events has little effect on these systems' detection results.

B. Maintain and Extend CPD

One limitation of CPD is that it relies on existing threat intelligence knowledge base (MITRE ATT&CK), which is subject to expansion. That is, the detection rule base in stage 1 and the indicator list in stage 3 of CPD need to be updated, if new attack techniques or behaviors emerge in the wild. Nonetheless, manually updating the rule base in stage 1 and the indicator list in stage 3 takes only a few minutes. Besides, we stress that unlike signature-based IDS relying on easily modifiable hard-coded strings, CPD is based on characteristic

attack behaviors and mostly uses immutable indicative strings in its stage 1. In other words, CPD depends on MITRE ATT&CK Matrix and high-level behavior rules that are subject to change in a much slower pace. By tracking changes made in MITRE ATT&CK Matrix, we find that MITRE updates the Matrix on a half-year basis. Our detection rule base and indicator list are based on a previous release (April 25, 2023). After inspecting the persistence tactic and techniques in the current release (April 23, 2024), we find that we do not need to update CPD at all.

C. Adaptability and Generality of CPD

CPD is based on MITRE ATT&CK Matrix, which is a general framework valued by organizations across the globe. As such, leading security vendors, on the one hand, continue contributing to this framework and, on the other hand, use this framework as a reference to develop detection rules/mechanisms. Besides, we use popular standard instrumentation-free logging frameworks for collecting system logs. Our CPD prototype interfaces with Elasticsearch, which is one of the most popular tools for event search and threat analysis among organizations. CPD is tested on a typical Windows Domain network (including both Windows and Linux machines as being monitored clients) as required in the MITRE emulation plans. CPD is deployed on a Linux machine functioning as a server that processes system logs shipped from client machines and performs attack detection. These characteristics, combined with its robust foundation in the MITRE ATT&CK framework, establish CPD as a versatile and comprehensive solution, readily deployable in a wide range of enterprise settings for effective persistence threat detection with minimal implementation effort.

D. Completeness of CPD

CPD's methodology, while illustrated through specific examples and MITRE techniques in the paper, embodies a comprehensive and generalizable framework for detecting persistence threats. The case-by-case analysis serves not merely as isolated instances but as representative samples of broader persistence threat patterns, showcasing CPD's practicality across varied threat landscapes. This approach ensures that while the examples may appear specific in the paper, the underlying principles – such as the segmentation into setup and execution phases – are universally applicable. Such a strategy underlines CPD's completeness, affirming its capability to address not just known scenarios but also to adapt and respond to emerging threats. CPD's effectiveness is further validated through extensive evaluation on diverse datasets, showcasing its superior attack detection rates and graph completeness compared to state-of-the-art methods

VIII. Related work

In Section I, we described the limitations of the existing threat detection system that CPD addresses, and complement the discussion on related work here:

Provenance-based IDS (PIDS). Learning-based PIDS use machine learning to model benign behavior from provenance graphs. They alert on deviations during runtime. Early models, such as [9], [20], detected anomalies at the graph level, complicating attack investigations. Newer models improve detection by focusing on time windows [17], edges [19], or nodes [18], [26], [27]. Despite advancements, these systems lack explanations for alerts, reducing interpretability. They also require extensive training data and slow down detection processes. Furthermore, they are vulnerable to concept drift and evasion attacks [28], [29]. Recent systems like [17], [18], [27] have addressed some issues but still fail to fully understand persistence attacks or include necessary contextual checks.

Heuristics-based PIDS, such as [13], [15], apply detection signatures on provenance graphs to transform them into high-level APT stage graphs. These systems calculate threat scores based partly on the rarity of events and the correlation of APT stages within the APT graphs. However, they face difficulties in linking fragmented APT attack stages due to not recognizing the dual nature of persistence attacks, often resulting in incomplete attack graph reconstructions. Consequently, they rank disconnected graphs so low that true attacks frequently remain uninvestigated. Moreover, these systems require benign training data to assign rarity scores to events and filter false alarms. Additionally, RapSheet [15] is designed as an offline detection system, leading to significant delays in threat identification and investigation. Unlike these systems, CPD excels in detecting persistence techniques by combining advanced detection rules with pseudo-edges and expert-guided edges. This unique approach enables CPD to effectively identify and investigate persistence threats in real-time, bridging the gaps that other PIDS fail to address.

Specialized Threat Detectors. Specialized threat detectors focusing on single stages of APT attacks are well-documented in the literature. For instance, Ho et al. [64] introduced the Hopper system, and King and Huang [65] developed the Euler system, both targeting lateral movement detection using network logs. There are also specialized systems for detecting phishing emails [66], data exfiltration [67], command and control (C2) activities [68], and ransomware [69]. To the best of our knowledge, CPD is the first specialized detector that focuses on persistence threats using provenance analytics, filling a critical gap in APT defense strategies.

Log Reduction Schemes. Numerous log reduction systems have been proposed recently, such as [70]–[74]. Unlike these systems, CPD specifically aims to reduce the reliance on IPC logs to enhance persistence detection, a focus not typically addressed by existing log reduction schemes. These systems, while complementary to CPD, can also be integrated to further decrease the size of audit logs.

IX. Conclusion

In this paper, we introduce CPD, a novel system dedicated to detecting persistence attacks. Distinctively, CPD leverages provenance analytics, moving beyond the basic detection rules traditionally used. This approach not only significantly reduces false alarms but also notably enhances accuracy in identifying genuine persistence attacks. Our evaluations, conducted on

both public datasets and datasets derived from rigorously executed MITRE emulation plans, demonstrate CPD's superiority over state-of-the-art detection methods. Furthermore, CPD incurs low runtime overhead, making it a valuable addition to the suite of threat detectors in enterprise settings.

REFERENCES

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

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

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

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

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

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

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

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

[9] X. Han, T. Pasqueir, A. Bates, J. Mickens, and M. Seltzer, "UNICORN: Runtime provenance-based detector for advanced persistent threats," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2020, pp. 1–18.

[10] S. Ma, X. Zhang, and D. Xu, "ProTracer: Towards practical provenance tracing by alternating between logging and tainting," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2016, pp. 1–15.

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

[12] W. U. Hassan, L. Mark, N. Aguse, A. Bates, and T. Moyer, "Towards scalable cluster auditing through grammatical inference over provenance graphs," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2018, pp. 1–15.

[13] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. N. Venkatakrishnan, "HOLMES: Real-time apt detection through correlation of suspicious information flows," in *Proc. IEEE Symp. Secur. Privacy*, 2019, pp. 1137–1152.

[14] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates, "NoDoze: Combatting threat alert fatigue with automated provenance triage," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2019, pp. 1–15.

[15] W. U. Hassan, A. Bates, and D. Marino, "Tactical provenance analysis for endpoint detection and response systems," in *Proc. IEEE Symp. Secur. Privacy*, 2020, pp. 1172–1189.

[16] M. N. Hossain, S. Sheikhi, and R. Sekar, "Combating dependence explosion in forensic analysis using alternative tag propagation semantics," in *Proc. IEEE Symp. Secur. Privacy*, 2020, pp. 1139–1155.

[17] Z. Cheng, Q. Lv, J. Liang, Y. Wang, D. Sun, T. Pasquier, and X. Han, "KAIROS: Practical intrusion detection and investigation using whole-system provenance," in *Proc. IEEE Symp. Secur. Privacy*, 2024, pp. 9–28.

[18] M. Rehman, H. Ahmadi, and W. Hassan, "Flash: A comprehensive approach to intrusion detection via provenance graph representation learning," in *Proc. IEEE Symp. Secur. Privacy*, 2024, pp. 142–161.

[19] J. Zeng, X. Wang, J. Liu, Y. Chen, Z. Liang, T.-S. Chua, and Z. L. Chua, "Shadewatcher: Recommendation-guided cyber threat analysis using system audit records," in *Proc. IEEE Symp. Secur. Privacy*, 2022, pp. 489–506.

[20] F. Yang, J. Xu, C. Xiong, Z. Li, and K. Zhang, "Prographer: An anomaly detection system based on provenance graph embedding," in *Proc. USENIX Secur. Symp.*, 2023, pp. 4355–4372.

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

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

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

[24] The MITRE Corporation. "MITRE T1547001." Accessed: May 2023, [Online]. Available: https://attack.mitre.org/techniques/T1547/001/.

[25] Q. Wang, W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, J. Rhee, Z. Chen, W. Cheng, C. A. Gunter, *et al.*, "You are what you do: Hunting stealthy malware via data provenance analysis," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2020, pp. 1–17.

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

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

[28] A. Goyal, X. Han, G. Wang, and A. Bates, "Sometimes, you aren't what you do: Mimicry attacks against provenance graph host intrusion detection systems," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2023, pp. 1–18.

[29] K. Mukherjee, J. Wiedemeier, T. Wang, J. Wei, F. Chen, M. Kim, M. Kantarcioglu, and K. Jee, "Evading Provenance-Based ML detectors with adversarial system actions," in *Proc. USENIX Secur. Symp.*, 2023, pp. 1199–1216.

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

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

[32] B. A. Alahmadi, L. Axon, and I. Martinovic, "99% false positives: A qualitative study of soc analysts' perspectives on security alarms," in *Proc. USENIX Secur. Symp.*, 2022, pp. 2783–2800.

[33] M. Wojtasiak, "The defenders' dilemma," 2023. [Online]. Available: https://info.vectra.ai/state-of-threat-detection.

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

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

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

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

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

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

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

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

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

[43] Red Canary. "Atomic Red Team." Accessed: Jan. 2023, [Online]. Available: https://atomicredteam.io/.

[44] Elastic NV. "EQL search." Accessed: Sept. 2023, [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/eql.html.

[45] R. Uetz, M. Herzog, L. Hackländer, S. Schwarz, and M. Henze, *You cannot escape me: Detecting evasions of SIEM rules in enterprise networks*, 2023. arXiv: 2311.10197 [cs.CR].

[46] A. Allievi, A. Ionescu, D. A. Solomon, K. Chase, and M. E. Russinovich, *Windows Internals, Part 2, 7th Edition*. Microsoft Press, 2022.

[47] The MITRE Corporation. "APT29." Accessed: April 2023, [Online]. Available: https://attack.mitre.org/groups/G0016/.

[48] The MITRE Corporation. "Wizard Spider." Accessed: April 2023, [Online]. Available: https://attack.mitre.org/groups/G0102/.

[49] S. White. "Windows Remote Management." Accessed: March 2023, [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/winrm/portal.

[50] The MITRE Corporation. "Carbanak." Accessed: April 2023, [Online]. Available: https://attack.mitre.org/groups/G0008/.

[51] Elastic NV. "Elasticsearch." Accessed: Sept. 2023, [Online]. Available: https://www.elastic.co/.

[52] NetworkX developers. "NetworkX." Accessed: Sept. 2023, [Online]. Available: https://networkx.org/.

[53] West Health Institute. "PyVis." Accessed: Sept. 2023, [Online]. Available: https://pyvis.readthedocs.io/en/latest/.

[54] D. Marshall. "NT Kernel Logger." Accessed: Feb. 2023, [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/devtest/nt-kernel-logger-trace-session.

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

[56] A. Riddle, K. Westfall, and A. Bates. "ATLASv2." Accessed: Oct. 2023, [Online]. Available: https://bitbucket.org/sts-lab/atlasv2/src/master/.

[57] VMware LLC. "Carbon Black Cloud." Accessed: Oct. 2023, [Online]. Available: https://www.vmware.com/products/carbon-black-cloud.html.

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

[59] The MITRE Corporation. "MITRE Engenuity Evaluation." Accessed: Jan. 2023, [Online]. Available: https://attackevals.mitre-engenuity.org/enterprise/wizard-spider-sandworm/.

[60] Elastic. "A Elastic rule sample." Accessed: Sept. 2023, [Online]. Available: https://github.com/elastic/detection-rules/blob/main/rules/windows/persistence_registry_uncommon.toml.

[61] SigmaHQ. "A Sigma rule sample." Accessed: Sept. 2023, [Online]. Available: https://github.com/SigmaHQ/sigma/blob/master/rules/windows/file/file_event/file_event_win_creation_unquoted_service_path.yml.

[62] Google Security Operations. "A Chronicle rule sample." Accessed: Sept. 2023, [Online]. Available: https://github.com/chronicle/detection-rules/blob/main/mitre_attack/T1053_005_windows_creation_of_scheduled_task.yaral.

[63] F. Pedregosa and P. Gervais. "Memory Profiler." Accessed: Oct. 2023, [Online]. Available: https://pypi.org/project/memory-profiler/.

[64] G. Ho, M. Dhiman, D. Akhawe, V. Paxson, S. Savage, G. M. Voelker, and D. A. Wagner, "Hopper: Modeling and detecting lateral movement," in *Proc. USENIX Secur. Symp.*, 2021, pp. 3093–3110.

[65] I. J. King and H. H. Huang, "Euler: Detecting network lateral movement via scalable temporal link prediction," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2022, pp. 1–16.

[66] G. Ho, A. Sharma, M. Javed, V. Paxson, and D. Wagner, "Detecting credential spearphishing in enterprise settings," in *Proc. USENIX Secur. Symp.*, 2017, pp. 469–485.

[67] Y. Ozery, A. Nadler, and A. Shabtai, "Information based heavy hitters for real-time dns data exfiltration detection," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2024, pp. 1–15.

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

[69] A. Kharaz, S. Arshad, C. Mulliner, W. Robertson, and E. Kirda, "UNVEIL: A Large-Scale, automated approach to detecting ransomware," in *Proc. USENIX Secur. Symp.*, 2016, pp. 757–772.

[70] M. A. Inam, Y. Chen, A. Goyal, J. Liu, J. Mink, N. Michael, S. Gaur, A. Bates, and W. U. Hassan, "Sok: History is a vast early warning system: Auditing the provenance of system intrusions," in *Proc. IEEE Symp. Secur. Privacy*, 2023, pp. 2620–2638.

[71] Z. Xu, Z. Wu, Z. Li, K. Jee, J. Rhee, X. Xiao, F. Xu, H. Wang, and G. Jiang, "High fidelity data reduction for big data security dependency analyses," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 504–516.

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

[73] Y. Tang, D. Li, Z. Li, M. Zhang, K. Jee, X. Xiao, Z. Wu, J. Rhee, F. Xu, and Q. Li, "Nodemerge: Template based efficient data reduction for big-data causality analysis," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 1324–1337.

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