APTHunter: Detecting Advanced Persistent Threats in Early Stages

MOUSTAFA MAHMOUD, Concordia University, Canada
MOHAMMAD MANNAN, Concordia University, Canada
AMR YOUSSEF, Concordia University, Canada

We propose APTHunter, a system for prompt detection of Advanced and Persistent Threats (APTs) in early stages. We provide an approach for representing the Indicators of Compromise (IOCs) that appear in the Cyber Threat Intelligence (CTI) reports and the relationships among them as provenance queries that capture the attacker's malicious behavior. We use the kernel audit log as a reliable source for system activities and develop an optimized whole system provenance graph that provides the causal relationships and information flows among system entities in a compact format. Then, we model the threat hunting as a behavior match problem by applying provenance queries to the optimized provenance graph to find any hits as indicators of an APT attack. We evaluate APTHunter on adversarial engagements from DARPA over different OS platforms, as well as real-world APT campaigns. Based on our experimental results, APTHunter promptly and reliably detects attack artifacts in early stages.

CCS Concepts: • **Security and privacy** → **Malware and its mitigation**; **Intrusion detection systems**.

Additional Key Words and Phrases: Threat intelligence, APT, attack detection

1 INTRODUCTION

Today's system intrusions are subtle and sophisticated. Using strategies of Advanced Persistent Threats (APTs), adversaries often launch targeted and stealthy cyber attacks against organizations to lurk into the enterprise network undetected for months or even years [65]. APT groups usually target sensitive and large corporations, including: government-based enterprises, e.g., BlackEnergy APT attacks against Ukraine power grids [38] and Ukrainian government websites defacement [39]; financial institutions, e.g., the Carbanak APT attack against banks around the world [22], the Chinese state-sponsored APT attack against Taiwan's financial trading sector [2], and Lazarus group attacks against cryptocurrency and blockchain industires [67]; and health care organizations, e.g., APT41, FIN4, and ORANGEWORM APTs target intellectual properties and personal information from health care services [11, 60, 72].

APT attacks behave differently than most regular malware instances. APT actors usually use native system tools that are used by system administrators, tricking the security controls to consider their activities as normal behavior [13] (see examples in [7]). In addition, the APT lifecyle consists of disparate stages that comprise multiple attack techniques, and the stages may spread across a long period of time [12, 64]. The MITRE ATT&CK framework [45] provides abstract definitions of tactics and techniques used by APT groups throughout the attack lifecycle [47]. Specific attack details are generally included in Cyber Threat Intelligence (CTI) reports, e.g., Indicators of Compromise (IOCs) and exploited vulnerabilities [24]. When a CTI report is released (primarily by anti-malware vendors) for a specific APT, enterprises may update their defense tools with the recent attack IOCs,

Authors' addresses: Moustafa Mahmoud, moustafa.mahmoud@mail.concordia.ca, Concordia University, Montreal, Canada; Mohammad Mannan, Concordia University, Montreal, Canada, m.mannan@concordia.ca; Amr Youssef, Concordia University, Montreal, Canada.

2576-5337/2022/9-ART $15.00
https://doi.org/10.1145/3559768

e.g., update antivirus signatures with hashes of the attack tools, block attacker's IP addresses in Firewall, and update IDS/IPS with vulnerability signatures.

Threat Hunting challenges. Once enterprise defense tools and controls are updated with the recent attack IOCs, they may be, then, able to detect suspicious events correlated with the attacker on enterprise hosts. Security Information and Event Management (SIEM) systems (e.g., Arcsight [6], ELK [21]) may correlate between events from different detection sources, and hence generate more mature alerts based on a predefined set of use cases (e.g., same malware detected in 10 workstations or more). These traditional tools are indeed crucial for protecting the network premises; however, they do not provide two pivotal features, making the security analyst blind to APT attacks: (a) understanding the causal relationship between different system events [44]; and (b) piecing together attack artifacts over a long period of time [3].

Most current threat hunting approaches also operate only on partial views of cyber attacks such as detection based on known IOCs, e.g., malware hash values, IP addresses of C&C, domain names owned/controlled by the attackers, file names used by the APTs [23, 58]; using heuristics [63], e.g., checking if a file appears to be a variant of known malware; or by using local behavior analysis [46]. The first method (known IOCs) can pinpoint exact known malware samples, but it misses mutated malware versions [70, 71]. The other two techniques may be able to detect mutated malware but often trigger a high rate of (costly) false positives [66]. Other threat hunting mechanisms include network system behavior analysis tools (e.g., [76]) and file system behavior analysis tools (e.g., [74]) which are based on detecting anomalies from the learned system normal behavior. However, they also generate an unmanageable number of false positives [9, 56], unless tuned with higher thresholds, which on the other hand, increase the probabilities of missing attacks.

Provenance Data Analysis. Provenance data analysis is a promising approach to tackle these APT-specific challenges; see e.g., [44, 51, 52]. System event logs are parsed into a whole system provenance graph which provides the causal dependency between system subjects (e.g., processes) and objects (e.g., files and sockets). Given all the artifacts of an attack, an analyst may find the root cause by issuing a backward tracing query on the provenance graph [35, 41, 43, 44].

Common provenance data analysis approaches assume that APT campaign stages occur in a short period of time (e.g., hours or days). Besides, they focus largely on detecting indicators of an APT attack as a whole (i.e., when the APT completes all its stages), rather than detecting individual malicious activities per every attack stage; hence, they cannot detect APT attack activities per stage (see e.g., [33, 50]), and thus unable to perform early-stage detection. HOLMES [51] provides rules for APT stages based on tactics, techniques, and procedures (TTPs) from the MITRE ATT&CK framework. The APT detection alert is generated based on the accumulative threat scores from all stage rules. HOLMES is found to be very accurate in APT attack detection, after all attack stages are completed; however, per-stage detection was not considered, and therefore per-stage results were unavailable.

We re-implemented a per-stage version of HOLMES, which we call S-HOLMES. Our experimental results show that S-HOLMES can accurately detect APT attacks from stage six (Complete Mission) onwards, but fails in the early stages. UNICORN [33] is an anomaly-based system (see also [77]) that generates alerts when the provenance of the whole system events deviates from the learned set of graphs representing benign activities; however, such anomaly-based systems are generally prone to high rate of false positives; as mentioned by the UNICORN authors, when normal behavior changes, UNICORN may generate false alerts.

Problem Statement. The main problem addressed in this paper is to promptly detect an ongoing APT campaign, which consists of several disparate stages over a long period of time, in early stages with high precision and sensitivity in real-time.

Our Approach and Contributions. APTHunter couples the attack semantics in the CTI reports and the audit log, to produce precise attack provenance queries. We complement the abstract definitions of the APT attack stage techniques (from MITRE ATT&CK framework) with the attack technical details available in the CTI reports to build the provenance queries.

During the runtime, APTHunter first normalizes the raw audit log records to a canonical form such that the relationships among the system entities can be causally tracked. The canonical log entries are then used to generate the whole system provenance graph. A significant point about our provenance graph is that it requires less memory compared to the log size on disk which facilitates real-time ingestion of events and generation of the graph over a long period of time. APTHunter then applies the attack provenance queries to the whole system provenance graph to generate alerts as per every APT attack stage.

We evaluate APTHunter on three different experimental setups. The first experiment is based on a dataset of red-team simulated APT attack campaigns generated by DARPA Transparent Computing program [73]. In the second experiment, we test APTHunter on recent real world APTs in our lab, and finally, we evaluate APTHunter's robustness against false positives with the two-week long DARPA benign dataset [73]. In summary, we make the following contributions:

- We propose APTHunter to detect APT attack activities per stage, including *early stages*, by filling the gap between the attack abstract definitions from the MITRE ATT&CK framework and the system low level event log, using the technical attack details from the CTI reports. We use the operating system kernel audit log to pinpoint the attack activities and provide root cause analysis of the attack evolution on the system for every attack stage. This is a clear shift from past works which mostly deal with APTs *as a whole*.
- As part of APTHunter, we design and implement a runtime audit log provenance tracking approach called LogCore. The output from this step is a normalized and compacted version of the audit log, ready to be used for further analysis by provenance graph mechanisms. Using LogCore, we build a whole system provenance graph based on the compact form of the kernel audit log.
- For deriving attack provenance queries for every attack stage from selected CTI reports, we design and implement a mechanism called ProvQuery. Our provenance queries represent attack behaviors rather than a set of specific IOCs, enabling us to detect mutated attacks in all stages, as attack behaviors often remain constant among mutations.
- During our evaluation, we test APTHunter on eight DARPA APT attack scenarios and two other recent real-world APT attacks to assess the performance of APTHunter in detecting attack artifacts in early stages. APTHunter consistently shows outstanding results in detecting APT attack artifacts as early as in Initial Compromise, with no false alerts when tested on the benign dataset. The compact log representation by APTHunter allows a compression of 93% for the provenance graph database generation. The average search time taken by APTHunter is 18% or 22% (based on APTHunter setting) of the average search time taken by S-HOLMES.

The source code of APTHunter, including LogCore and ProvQuery, and our generated provenance queries are available at https://github.com/APT-Hunter.

2 ATTACK STAGES AND THREAT MODEL

In this section, we describe an attack scenario carried out by the DARPA red-team during their recent engagement 5 [73], and use it to describe different APT attack stages.

2.1 APT Attack Stages

In one of the DARPA simulated attack scenarios, a vulnerable remote web server (*Nginx*) running on *FreeBSD* is exploited which allows the attacker to have remote access to the victim system. In the following, we disassemble this attack scenario into seven different APT stages following previous work [51] and industrial technical reports

on APT attacks kill-chain [8, 26, 75]. The description of every stage contains attack artifacts and technical details for each stage. Our main goal is to pinpoint these artifacts to be able to detect different attack stages.
Initial Compromise. The attacker targets the *Nginx* web server by sending a malformed HTTP POST on port 80 to exploit a remote code execution vulnerability. The malicious shellcode is added successfully to the executable region of the memory. Next, the attacker gains control over the *Nginx* process.
Establish Foothold. The shell code executes on the target, granting the attacker access to the victim device through a reverse shell. Now commands can be communicated between the victim device and the Command and Control (C2) server.
Escalate Privileges. The attacker exploits a kernel vulnerability to grant the *Nginx* process super user (root) privileges.
Internal Reconnaissance. The attacker uses the implemented shell to send several commands to collect information about the internal system. Commands include `hostname`, `whoami`, and `cat /etc/passwd`.
Lateral Movement. During this attack scenario, the attacker does not connect laterally to any other device in the victim network. However, an example of artifacts in Lateral Movement stage is when an attacker finds credentials in clear text on the victim device, and then uses those to remotely connect to other devices in the internal victim network.
Complete Mission. The attacker exfiltrates sensitive files (/etc/passwd and /etc/shadow) to an external device.
Cleanup Tracks. The attacker clears traces by removing the created temporary files and clearing the system log files.

Note that APT stages are not necessarily executed sequentially by the attackers. They are based on the attack circumstance and the victim environment. We observed this based on the analysis of several CTI reports for different APTs (e.g., see [19]) in the wild. For example, attackers may escalate privilege directly after getting limited access (Initial Compromise) to the victim network. In other circumstances, attackers may need to conduct intensive Internal Reconnaissance to search for further vulnerabilities, and/or security loopholes in the victim system for privilege escalation, and be able to move laterally to connect to sensitive victim devices.

2.2 Threat Model

APTHunter aims to detect APT attacks which exploit application and service vulnerabilities to get into victims systems all the way to data exfiltration and cleanup tracks. APTHunter relies on kernel audit logs as sources for system events. We assume that the kernel space and the audit log data collected from it are part of the trusted computing base (TCB). Any kernel-level attacks that target the integrity of the audit log are beyond the scope of this paper. Other solutions such as HACK [48] (a Hardware-Assisted Kernel Compartmentalization) can be used to prevent kernel-based attacks. Side channel attacks (e.g., power-analysis attacks, timing attacks) are out of scope.

3 APTHUNTER DESIGN

The main idea of APTHunter is to couple the high level semantics in the CTI reports and the low level details in the audit log to pinpoint attack artifacts in early attack stages; for a high-level overview see Figure 1. In the following, we provide a description of different components of APTHunter.

3.1 Stream Processor

Our system takes as an input the kernel audit log retrieved from different hosts that may run different operating systems. Kernel logs can be monitored and retrieved by using, e.g., Event Tracing for Windows (ETW),[1] Audit

[1]https://docs.microsoft.com/en-us/windows/win32/etw/event-tracing-portal

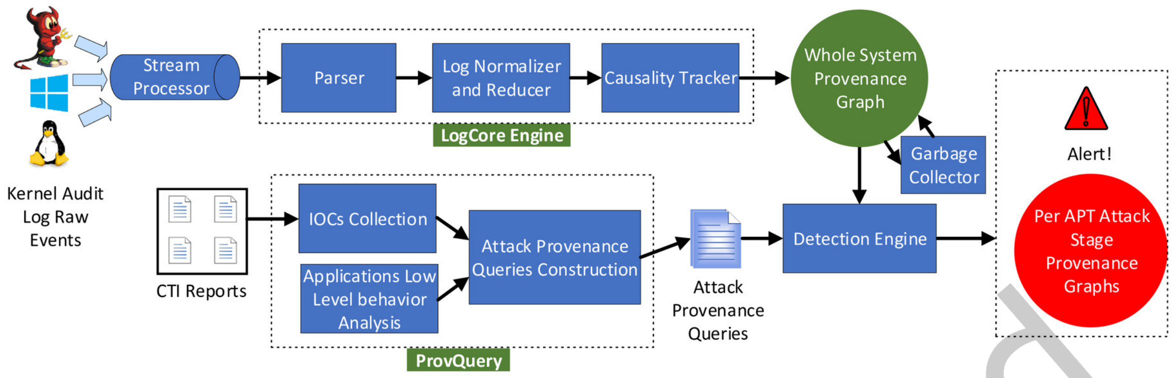
Stream Processor
Kernel Audit Log Raw Events
Parser
Log Normalizer and Reducer
Causality Tracker
LogCore Engine
Whole System Provenance Graph
Garbage Collector
CTI Reports
IOCs Collection
Applications Low Level behavior Analysis
Attack Provenance Queries Construction
ProvQuery
Attack Provenance Queries
Detection Engine
Alert!
Per APT Attack Stage Provenance Graphs

Fig. 1. An overview of APTHunter's layers of abstraction from raw system events to attack discovery.

daemon (Auditd) for Linux,[2] and DTrace for FreeBSD.[3] This input captures operations related to different system entities including processes (e.g., execution, cloning, and privilege change), and files (e.g., file creation, deletion, ownership and permission changes).

Audit records from different operating systems are published to a stream processing server (*Kafka* [5]). Endpoints are configured to send the audit records to a specific Kafka topic based on the operating system hosted on the endpoint. APTHunter proceeds by consuming records from the streaming server to the APTHunter LogCore engine. In our deployment, all APTHunter's components run on the same host, but they also can run using different deployment strategies (e.g., virtual machines, containers, dockers). To onboard new computers to APTHunter, they need to be configured to send the event log to the stream processing server.

3.2 LogCore Engine

The audit log contains an immense number of raw low level system events that do not readily show causal relationships between system entities, and hence cannot be used to pinpoint high level attack artifacts. Our LogCore engine aims to provide a canonical representation of the audit log that is compact and symmetric across different system events. Given the low level kernel audit log, system subjects (e.g., processes) and objects (e.g., files and sockets) are uniquely identified, and system events are extracted and manipulated to facilitate provenance root cause analysis. The resultant canonical form models the system entities and their relationships as a labeled, typed, and directed graph, called whole system provenance graph, representing the runtime status of the system. In this graph, nodes represent system entities (e.g., processes, files), while edges represent the relationships among those nodes. These relationships enclose the information flow and direction, and the causality among those system entities.

Our whole system provenance graph creation differs from previous work (cf. [29, 30, 35, 40]). While the previous work focused mainly on the causality tracking part when generating the provenance graph, our LogCore engine processes the raw kernel event logs before generating the graph. The processing includes generating a uniformly structured and reduced event log representation form with system objects uniquely identified in the whole system runtime, and causal relationship between system entities preserved. This is a crucial requirement in our design to facilitate coupling the whole system graph (LogCore output) with high-level attack artifacts from the CTI reports.

Technical details of the LogCore engine including generating the canonical form representation and the causality tracking are provided in Appendix A.

[2]https://man7.org/linux/man-pages/man8/auditd.8.html
[3]https://wiki.freebsd.org/DTrace

3.3 Attack Artifacts Construction

We design ProvQuery to fill in the semantic gap between the abstract APT attack definitions in the MITRE ATT&CK framework and the system event log using the technical attack details from the CTI reports. We manually extract IOCs and the attacker's behavior from the threat intelligence reports by industry (e.g., Darktrace [15] and FireEye [25, 28]). CTI reports describe the attack tactics (e.g., Lateral Movement) and techniques (e.g., Exploitation of Remote Services) as well as the attacker's behavior [36].

Threat intelligence information is available in public and private threat intelligence feeds (e.g., AlienVault [4], Abuse.ch [1], EclecticIQ [20]), as well as from threat intelligence reports by industry (e.g., FireEye [27], Red Canary [64]). The attack artifacts are described in structured and semi-structured formats including, OpenIOC,[4] Structured Threat Information eXpression (STIX),[5] Cyber Observable eXpression (CybOX),[6] YARA,[7] etc. Given the rapid increase in the attack volume and sophistication, the attack artifacts are often descried in unstructured text (as in the CTI reports by industry).

We model the behavior appearing in the CTI reports as a set of attack provenance queries using our proposed ProvQuery approach. Every APT attack stage is represented by a set of provenance queries that reflect the causal relationship between the original source of the event and the attack event. To facilitate building the attack queries, we need to understand the low level behavior of popular applications, and determine that the low level attack activities in the event log can be located by ProvQuery based on the high level CTI reports.

Finally, we build the provenance queries on top of Neo4j [57] graph database engine to leverage its mature native graph database storage and processing. Our system synthesizes Neo4j queries to match attack artifacts from the stream of system events, and perform stateful computation and attack graph construction which represents the temporal evolution of the attack events.

Listing 1 Single process application example: event log records for a single process application (scp) which handles tasks sequentially (i.e., does not spawn new process to handle new tasks).

(a) Scp Process Execve SYSCALL

```
type=SYSCALL TS=1618072037.116 ID=827810
syscall=execve exit=0 ppid=7162 pid=7176
exe="/usr/bin/scp"
type=EXECVE a0="scp" a1="/etc/hosts"
a2="x10@192.168.8.134:~/victim_data/hosts"
```

(b) Close SYSCALL

```
type=SYSCALL TS=1618072037.116 ID=827812
syscall=close ppid=7162 pid=7176
exe="/usr/bin/scp"
type=PROCTITLE TS=1618072037.116 ID=827812
proctitle="bash"
```

Low Level Behavior of Popular Linux Applications. We conduct a manual study of five Linux applications that are widely used in APT attacks: Firefox, Nginx, SSH, Apache and PHP. We select these applications based on the DARPA Engegement 3 and 5 reports [17, 18], the APT41 FirEeye CTI report [25], and APT35 CTI report [28].

[4]https://github.com/mandiant/OpenIOC_1.1
[5]https://stixproject.github.io/
[6]https://cyboxproject.github.io/
[7]http://virustotal.github.io/yara/

We study the attack artifacts from available aforementioned CTI reports and the audit log to validate the following: (1) if we can map the high level artifacts in the CTI reports to the low level audit log; (2) how application tasks are handled in the low level audit log. We find that behavior of applications can be different based on the type of the application. For instance, server applications that need privilege separation, fork processes if the applications will start new tasks (see Figure 2). On the other hand, applications that are based on a single process such as wget and scp, do not fork a child process, but rather, they handle tasks sequentially one by one. See Listing 1 for an event log record example for single process applications. This observation is pivotal in our design to bridge the semantic gap between the high level attack artifacts in the CTI reports and the low level audit log. We used that in the design of the provenance queries.

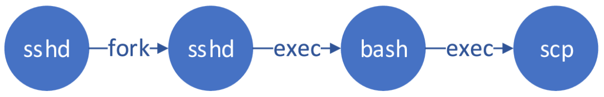

Fig. 2. The sshd daemon process forks a child process and the child process further spawns other process for various functionalities (secure copy).

Attack Provenance Query. From CTI reports, we manually extract attack artifacts including accessed files and executed processes. We model the attack behavior from the CTI report as a set of attack provenance queries using ProvQuery, which provides explicit constructs to specify system entities, events, as well as event relationships. This facilitates the specification of rules to detect known attack behaviors or enterprise-wide security policies. ProvQuery constructs attack queries by defining seven main components: source subject, target object, immediate syscall, intermediate process, intermediate syscall, and process tree path length. The queries are in the form:

$$\langle N_1 : node\rangle\langle R_I : syscall\rangle\langle length\rangle\langle N_2 : node\rangle \qquad \text{(Prerequisite Query)}$$

$$\langle N_3 : node\rangle\langle R_M : syscall\rangle\langle N_4 : node\rangle \qquad \text{(Main Query)}$$

where nodes N_1 and N_4 denote the source subject and target object respectively, while nodes N_2 and N_3 denote the intermediate processes. Subject examples include processes such as the `Firefox` process, while object examples include files such as `/etc/passwd`. R_I and R_M denote intermediate and immediate relationships (syscalls) respectively. Syscall examples include `read`, `write`, `execve`, and `connect`. *length* denotes the process tree path length which limits the collection of information flows starting from node N_1 to all successor nodes N_2 with the maximum process tree path equals to the specified *length*. Figure 2 shows `sshd` daemon (N_1) has relationships of `fork` and `execve` with process `scp` within a path of $length = 3$. Note that the process tree path length (*length*) can be adjusted by the enterprise based on the needed level of visibility, given that increasing it would increase the processing time, and possibly the number of false positives (see Section 5).

Prerequisite Query's goal is to find artifacts from the event log which are prerequisites for another set of activities (represented by Main Query) that may occur in the far or near future. The main query is constructed based on attack activity in the CTI reports and is conditioned by matches from the prerequisite query. Coupling together the matches from both queries (each with different triggering point in time), would indicate with high certainty the occurrence of an attack.

The prerequisite and main queries are then used to produce the provenance query that represents the attack behavior. Technically, the main query contains conditions which match certain activities corresponding to a specific attack stage from the CTI report. Example of this includes using well-known OS commands (e.g., `whoami`, `hostname`, and `id`) as part of Internal Reconnaissance. We consider signals generated from the main query as weak signals that may indicate occurrence of attack activities. However, we do not rely only on this to generate an alert. In addition, we also add the prerequisite query as a factor of certainty, if matched, would indicate with high probability that the activities are part of attack. Note that the prerequisite query is optional in our design, and thus omitted when no prerequisite conditions are needed for the main query. In Section 4, we show representative examples of the provenance queries as per every attack stage.

Query Extraction Sources. To construct attack queries in the ProvQuery approach, first, we manually locate different stages in the CTI report. Second, we identify different arguments and convert them into generic forms to facilitate covering mutated attack artifacts. Then, we construct attack queries based on conditions per every stage. We use the following two excerpts from DARPA engagement 5 ground truth report available in [18], to show, step by step, how the attack provenance queries are constructed. These two attack scenarios are also included in the list of APT attack scenarios used in our evaluation. The first excerpt shows part of the *Firefox Drakon APT* attack scenario using a compromised domain.

> Exploit Firefox by browsing to the hijacked www.yale.edu. This resulted in C2 connections to 35.106.122.76:80 and 69.155.209.87:80. The attacker used elevate to gain root privileges. The attacker got the processing listing, found the sshd process, and injected into it using a new process injection technique. This resulted in new C2 connections to aforementioned addresses.

The second excerpt shows part of *Nginx Drakon APT* attack scenario exploiting public-facing application (Nginx).

> Exploit Nginx by simulation of remote code execution on the listening port of the webserver TCP 80. The malformed HTTP POST was sent from 128.55.12.167 and resulted in C2 to 4.21.51.250:80. The attacker got the hostname and username.

Stage Identifier. To construct attack queries, we first locate attack artifacts in the CTI report. Then, we identify the attack technique used and map it to the corresponding attack stage as defined in MITRE ATT&CK. In the first excerpt, for instance, the first sentence describes Initial Compromise stage as per MITRE ATT&CK T1584.001 Compromise Infrastructure – Domains, and then the second sentence denotes the Establish Foothold stage. The next sentence shows an example of the Escalate Privileges stage, and the sentence after describes activities in the Internal Reconnaissance stage.

Robust Identification. Attackers often mutate attack artifacts to mislead defense systems; e.g., they may use different IP addresses, files with different hash values. To detect mutated artifacts, we transform identified objects (e.g., IP address) into a generic form. The above excerpts mention several IP addresses, which we simply map into %UnTrustedIPAddresseses%. Similarly, we replace Firefox with %BrowserProcess%. Hence the resulting rule is not only valid for Firefox and those IP Addresses but it is also valid for other browsers and other untrusted IP Addresses. The list of %UnTrustedIPAddresseses% includes all external IP Addresses that are not whitelisted enterprise-wide. Whitelisted IP Addresses may include IP ranges of vendors, suppliers, and third parties that the enterprise has direct business relationship with, as well as IP addresses for well-known corporations (e.g., Google, Microsoft). Network orchestration solutions (e.g., Tufin) may be used to populate and update this list on time. In our experiments, we considered IP addresses in the private subnet ranges as trusted, while assuming that all other IP addresses are untrusted. The system also includes definitions of the browser processes with a default set of known browsers, and can also be configured by system admins.

3.4 Searching for Indicators of Attack

Finally, we model threat hunting as determining whether the detection engine generates hits by applying different constructed attack provenance queries to the whole system provenance graph. A match indicates that the attack behavior manifests itself inside the whole system provenance graph.

Constructed provenance queries represent different attack signals that generate alerts if certain crafted criteria matched. They provide real-time monitoring of endpoint events for malicious behaviors and methodologies of attack. Unlike specific set of IOCs (e.g., hash values, IP addresses), provenance queries are evergreen heuristics,[8] and hence are perpetually relevant. Provenance queries identify malicious activities by searching for behaviors rather than specific indicator values.

We note at this point that the main query parts of the provenance queries, which are built based on the CTI reports, express the high-level flows between system entities. In contrast, the whole system provenance graph, even if it is based on the carefully crafted canonical audit log representation, still represents the low-level activities of the system. As a result, locating an activity in the provenance graph based only on the main query is challenging, as an edge in the main query might correspond to a path in the whole system provenance graph. An example of that is when a compromised process (e.g., *Nginx*) forks another instance of itself before reading a confidential file. We observe this kind of behavior for server-based processes, when treating new tasks. Also attackers often create this kind of behavior to add noise to their activities to escape detection. Therefore, we use the prerequisite query technique that can help in matching a single edge in the main query to paths in the whole system provenance graph. In the prerequisite query, we use the intermediate path (R_I) with a process tree path length ($length$) to fill in that gap.

A common practice in attack forensics to determine the process tree path length is to do backward tracing from the artifact matching point to reach an initial compromise point [41, 43, 44]. Incremental matching is another approach used in the literature [10, 51]. In incremental matching, the results of the previously matched artifact, that may be a prerequisite for another artifact, is stored and propagated to all the low level entities that have dependencies on the entities of that matched artifact. An example of this, if initial compromise artifact is detected on node (N_1), this information will be propagated to all child nodes that are dependant on the node (N_1). Unfortunately, both techniques are computationally expensive in real-time enterprise setting as the provenance graph grows with the time to include millions of activities spanned across a long period of time.

Instead, we tag nodes with matched artifacts that are prerequisites for other artifacts. So now, to determine if artifacts related to certain attack (e.g., *Credential Dump*) match, we issue a backward tracing to processes previously tagged with the prerequisite conditions. For the *Credential Dump* example, we search for processes tagged as *Compromised* or *Super User Privilege.* Hence the $length$ represents the path length between the processes tagged with the matched prerequisite conditions and the processes with main query.

APTHunter comes with a preconfigured set of provenance queries to detect different attack stages. The provenance queries are constructed based on the available CTI reports for the selected set of APT attacks with the goal to detect matched or similar attack behavior for other APTs. The construction of provenance queries is an offline stage in our system. Consuming and processing the event logs, generating the whole system provenance graph, executing the provenance queries, and generating attack detection alerts occur at runtime. Security engineers can also write their own provenance queries to detect newly published APT attack artifacts.

When a provenance query triggers an alert, and an event matches in the whole system provenance graph, the resulting alert shows the context of the match. The context is represented by a graph showing the evolution of events between different system entities involved in the attack behavior.

3.5 Garbage Collector

We design a garbage collector (GC) which aims to allow APTHunter to efficiently use the whole system memory and promptly get results from the execution of the provenance queries. The idea here is to only keep in the provenance graph database and hence in memory all entities which are eligible to be queried by the ProvQuery and discard all those which are not eligible. An entity is eligible when it is within the selected process tree path

[8]https://blog.juriba.com/evergreen-it-concept-or-reality

length. The garbage collector has two processes; mark and sweep. The mark process marks all eligible entities as "live". When a process terminates, the tag "live" will be removed, unless the process is within the selected process tree path length. In the latter case, the tag will be kept. All direct entities which are not processes and which have relationships with "live" entities are also marked as "live". We make sure that no node will be removed even if it is exceeded the path length unless it is terminated. We also make sure that terminated processes are covered when they are within the process path length. The sweep process then scans all entities and deletes those without the tag "live" to reclaim memory.

4 PER STAGE PROVENANCE QUERY CONSTRUCTION

In this section, we provide representative examples of attack provenance queries to detect different APT attack stages. We use the attack excerpts from Section 3.3 in the text below.

4.1 Initial Compromise

The first excerpt mentions several entities and actions. We transform them into provenance query conditions to pinpoint the attack stage. For instance, the first sentence clearly denotes a %BrowserProcess% connects to a hijacked website (%TrustedIPAddresses%), and this results in connection to (%UnTrustedIPAddress%) under the attacker's control. This is clearly a *Domain Hijaking (T1584-001)* attack technique as in MITRE ATT&CK framework.

Hence, we identify the prerequisite query to be as follows:
Find all processes N_2 with the following conditions:

$$N_1 \in \{BrowserProcesses\} \wedge \ R_I = fork \ \wedge \ N_2 \in \{RemoteAccessProcesses\}$$

where *RemoteAccessProcesses* is the list of processes used for remote access (e.g., SSHD). Now, the main query aims to find all hits with the following conditions:

$$N_3 \in \{N_2\} \ \wedge R_M = connect \ \wedge N_4.ip \notin \{TrustedIPAddresses\}$$

In the second excerpt, it is clear that a reverse shell has been sent to the attacker's C2 server, after successfully exploiting a buffer overflow in a vulnerable Internet facing service (*Nginx*). We correlate this technique with MITRE ATT&CK *T1190 Exploit Public-Facing Applications.* This correlation is crucial as the same MITRE ATT&CK technique, used in different CTI reports, describes the same attack behavior, and hence the same provenance query can be leveraged to detect it. We translate the attack behavior as a provenance query for reverse shell detection with the following criteria:
The prerequisite query:

$$N_1 \in \{PublicFacingProcesses\} \ \wedge R_I = accept \ \wedge N_2.ip \notin \{TrustedIPAddresses\}$$

The main query:

$$N_3 \in \{PublicFacingProcesses\} \ \wedge \ R_M = connect \ \wedge$$
$$N_4.ip \notin \{TrustedIPAddresses\} \ \wedge N_4 \notin \{N_2\}$$

where N_2 and N_4 represent sockets (IP Address, Port).

The reverse shell rule aims to detect untrusted connections, where no previous connections are received from the same pair of IP address and port. This may indicate that a remote attacker is trying to exploit a vulnerable public-facing application with open ports to receive a reverse shell to an attacker's C2 server. Internet-facing applications include web applications, databases (e.g., MySQL), standard services (e.g., SMB and SSH), web servers (e.g., Apache and Nignx), and any other applications with Internet accessible open ports [54].

Similarly, other Initial Compromise techniques from MITRE ATT&CK are correlated with matched artifacts from the CTI reports and hence mapped into provenance queries that will be able to identify the activity in event logs. As mentioned in Section 3, the prerequisite query is optional in our design. For example, the provenance query for MITRE ATT&CK *T1571 Non Standard Port* technique is identified by the main query. The aim here is to find all hits for process N_3 with the following conditions:

$$\begin{aligned} N_4.ip \notin \{TrustedIPAddresses\} \wedge \\ Pair(Process\ N_3\ and\ Port\ N_4.port) \notin \\ \{ServicePortList\}\ \wedge\ R_M = connect \end{aligned}$$

Adversaries may make changes to the standard port used by a service to bypass filtering controls. For example, using SSH over port 2222 or port 587 as opposed to the traditional port 22. Another form of *T1571 Non Standard Port* technique is by using a port that is typically used by a well-known service with another service, e.g., using port 88 with processes other than the *lsass.exe*

in Windows. This may indicate *kerberoasting* [61] activity in which adversaries attempt to dump *lsass* memory for credential stealing. This is a well-known technique used by several sophisticated APTs [55].

4.2 Establish Foothold

Once the adversaries successfully compromised the system, they would seek for a shell. We formalize the conditions for the prerequisite query to be as follows.
Find all processes N_2 where:

$$\begin{aligned} &N_1 \in \{CompromisedProcesses\} \wedge \\ &R_I \in \{fork, execute\} \wedge \\ &length \leq SelectedLength \end{aligned}$$

CompromisedProcesses is a set of all processes tagged as compromised from the Initial Compromise stage. We then use the prerequisite query to list all processes forked or executed by those compromised processes up to the max process tree path length ($SelectedLength$) tuned by the security engineer. N_2 holds the list of those resultant processes, which will be used in the main query. Now for the main query, the aim is to find all processes N_3 where:

$$\begin{aligned} &N_3 \in \{N_2\}\ \wedge\ R_M \in \{fork, execute\} \wedge \\ &N_4 \in \{CommandLineUtilities\} \end{aligned}$$

4.3 Other Attack Stages

In the following, we discuss the logic behind detection of malicious behaviours of the next APT attack stages. We also provide representative examples of provenance query conditions with the full list consolidated in Appendix B.
Escalate Privileges. As in the first excerpt, “The attacker used elevate to gain root privileges.” This indicates that the attacker exploited a vulnerable service on the device which resulted in giving the attacker root access. Another way of escalating privilege is by executing commands using privileges of super user account. Adversaries may also abuse the *scheduled task* functionality in an OS to facilitate privilege elevation.

Furthermore, adversaries may attempt to dump credentials to escalate privilege, to laterally move to another more sensitive device in the network, and/or to extract sensitive information. Adversaries may also compromise valid domain accounts, allowing access to privileged resources in the domain (T1078.002: Valid Domain Accounts). Here, we provide the provenance query conditions for *scheduled task* and *Valid Domain Accounts* attack techniques. The query conditions for the other attack techniques are provided in the GitHub repository.

The prerequisite query for the *Scheduled Task* attack technique:

$$N_1 \in \{cron.d\} \ \wedge R_I \in \{fork, execute\} \ \wedge length \leq SelectedLength$$

The main query:

$$N_3 \in \{N_2\} \ \wedge \ R_M \in \{chown\} \ \wedge N_4.uid \in \{SuperUsers\}$$

The prerequisite query for the *Valid Domain Accounts* attack technique:

$$N_1.uid \notin \{DomainUsers\} \ \wedge \ N_2 \in \{ScriptingProcesses\}$$
$$\wedge \ R_I \in \{fork, execute\} \ \wedge \ length \leq SelectedLength$$

The main query:

$$\begin{aligned} &N_3 \in \{N_2\} \ \wedge \ N_3.uid \in \{DomainUsers\} \wedge \\ &R_M = connect \ \wedge \ N_4 \in \{InternalIPAddresses\} \wedge \\ &N_4.ip \in \{DomainIPAddresses\} \end{aligned}$$

Internal Reconnaissance. The adversaries now try to discover the internal network and the enterprise assets. They can use the operating system's native tools (e.g., `whoami`, `hostname`), and they may use OS utilities e.g., PowerShell, and customised tools e.g., Remote Access Trojans (RATs). Adversaries may also conduct port scanning to discover the vulnerable internal services. We create four provenance queries to detect those cases as in Appendix B.

Lateral Movement. Following their objective to find the key assets in the enterprise network, adversaries may explore the network by connecting to other internal devices. They may use techniques including *Pass the Hash*[9] after stealing credentials. We formalize the provenance queries to detect connections to internal devices from any host in which initial compromise or internal reconnaissance activities were detected.

Complete Mission. In the Complete Mission stage, attackers now have persistent access to the victim network. They lurked internally in the network and identified sensitive assets. They now decide to complete the mission by exfiltrating sensitive information and/or destroy the system altogether. In the following, we formalize provenance queries to detect several attack techniques in this APT stage.

Exfiltration Over C2 Channel. Once the adversaries have identified and collected the data from the enterprise, they try to get it out of the enterprise network by exfiltrating it over existing command and control channel (T1041: Exfiltration Over C2 Channel). Two prerequisite conditions should be satisfied before the attackers would be able to exfiltrate the data. They should have compromised the system and identified the sensitive assets (e.g., directories, files) that they would then steal or destroy. These two conditions may not be tied together as the adversary may have accessed the system using stolen credentials. Thus, sending data outside of the enterprise network to a non-trusted IP address after either compromising the system or identifying sensitive assets, would trigger an alert of sensitive data leakage by the provenance query.

Exfiltration by Bypassing Defense Controls. Enterprises may use several data loss prevention (DLP [37]) controls to detect and prevent such data leakage based on a predefined set of criteria including size of the data (e.g., > 10MB), or the frequency of the exfiltration events even if the outbound data size is less than the threshold (e.g., more than 10 connections with size of > 1MB from the same user).

Adversaries may want to stop software security agents (used by those controls to communicate with the DLP systems) installed on the target devices before being able to exfiltrate data. Otherwise, the data exfiltration will be failed, and indeed, security analysts would receive alerts about the attempts. Running and stopping services on devices, usually need the super user privilege. Thus, adversaries may leverage privilege escalation to have

[9]https://attack.mitre.org/techniques/T1550/002/

super user access in the device and then stop the agent (or reinstall a compromised version of it), before sending data outside the network.

Destroy System. Here the adversaries aim to either damage the file system or to remove sensitive enterprise information after they have successfully exfiltrated them. We crafted multiple provenance queries to detect those Complete Mission activities.

Cleanup Tracks. Provenance queries to detect suspicious *File Deletion*, *Remove Log Files*, and *Clear Log Commands* activities are created to pinpoint such suspicious behaviors.

5 EXPERIMENTAL EVALUATION

We configured APTHunter with the provenance queries consolidated in Appendix B.

We set process tree path length ($length = 3$) for our evaluation. We examined different $length$ values spanning from 1 to 9, and found that $length = 3$ provides the optimum results for precision and recall values with plausible attack search time.

We evaluate APTHunter's efficiency and accuracy in three different experiments. In the first experiment, we use a set of DARPA Transparent Computing (TC) program red-team adversarial engagement scenarios which are set up simulating an enterprise network. The setup contains target hosts (BSD, and Linux) with the kernel audit log enabled. During the engagement period, benign background activities were continuously run in parallel to the attacks from the red team. In the second experiment, we further test APTHunter on real-world APTs whose CTI reports are publicly available. To reproduce the attacks described in the public threat reports, we obtained and executed their binary samples in a controlled environment and collected the kernel audit logs from which we build the whole system provenance graph database. In the third experiment, we evaluate APTHunter's robustness against false alerts in benign dataset.

Also, to show the effectiveness of APTHunter, we compare APTHunter results with the previous work (HOLMES [51]) as it is a APT detection rule-based system, although its goal was different from ours (detecting APT as whole vs. detecting APT per stage). Since we were not able to acquire the implementation of HOLMES, we reimplemented a per stage version of it, and call it S-HOLMES. We validated the correctness of this reimplementation by comparing the results of S-HOLMES with the published results, besides verifying with the authors of HOLMES.

5.1 Evaluation on the DARPA TC Dataset

This experiment was conducted on a dataset released by the DARPA TC program [16], generated during the red-team adversarial engagements 3 and 5 conducted in April 2018 [17] and May 2019 [18], respectively. In the engagements, different services were set up, including web servers, SSH servers, email servers, and SMB servers. An extensive number of benign activities were running, including system administration tasks, web browsing, downloading, compiling, and installing multiple tools. To execute the attack scenarios, the red-team relies on threat descriptions, which we use to construct provenance queries for each stage for all attack scenarios. These threat descriptions are included in the ground truth reports provided by DARPA. We use the ground truth reports as the CTI reports for DARPA red-team activities.

We evaluated APTHunter on eight attack scenarios on FreeBSD, Ubuntu 12.04 (64 bit), and Ubuntu 14.04 (64 bit). The attack scenarios cover several attack techniques per every stage. Techniques including Drive-by Compromise (e.g., using a vulnerable version of Firefox), and Exploit Public-Facing Application (e.g., vulnerable Nginx) are examples of attack vectors used for Initial Compromise. In Escalate Privileges, adversaries are trying to gain higher-level access. Techniques including `setuid` and `setgid`, and `sudo` and `sudo caching` are examples of techniques used to elevate privilege in DARPA. See Table 1 for a summary of these techniques for the DARPA dataset streams.

Stream No.	Duration	Platform	Scenario No.	Initial Access Technique	Attack Surface
1	0d1h17m	Ubuntu 14.04 (64bit)	1	Drive-by download	Firefox 42.0
2	2d5h8m	Ubuntu 12.04 (64bit)	2	Drive-by download	Firefox 42.0 / Trojan / RAT
3	1d7h25m	Ubuntu 12.04 (64bit)	3	Drive-by download	Firefox 42.0 / Trojan / RAT
4	2d5h17m	FreeBSD 11.0 (64bit)	4	Web shell	Web shell / Nginx backdoor / RAT
5	8d7h15m	FreeBSD 11.0 (64bit)	5	Web shell	Nginx backdoor / sudo
			6	Public-facing	Nginx backdoor
6	0d0h36	Ubuntu 14.04 (64bit)	7	Drive-by download	Firefox 42.0 / sudo /sshd / Process inject
7	1d22h58m	Ubuntu 12.04 (64bit)	8	Drive-by download	Firefox 42.0 / sudo /sshd / Process inject

Table 1. Datasets. Streams 1 to 5 are from DARPA Engagement 3, Streams 6 and 7 are from DARPA Engagement 5. Streams 5 contains two independent attack vectors occurring on the same host.

Setup. To facilitate comparing detection results with S-HOLMES, we calculated the threat and the benign scores per every attack stage rather than for the whole campaign. To determine the score optimal threshold value that can be used to generate attack alerts, we measure precision, recall, and F-score by varying threshold values until we achieve maximum precision and recall values. As a noise reduction factor, we also implemented S-HOLMES using $path\_factor = 3$ as recommended by the authors. We do not take noise reduction rules into consideration, as they are solely based on learning the system benign behavior. Since the benign behavior varies with time and based on the circumstance (e.g., on system patching, rebooting etc.), we believe it will have little impact on the results. We then compare the generated alerts with the precise ground truth provided by DARPA to calculate the precision (as indicator of false positives), the recall (as indicator of false negatives), and the F-score (the harmonic mean of precision and recall).

Detection Results. Table 4 shows the results for S-HOLMES and APTHunter for the eight DARPA attack scenarios in terms of precision, recall, and F-score values.

Table 2 and Table 3 show the measurements of S-HOLMES in terms of threat and benign scores calculated as per every attack stage for DARPA attack scenarios and the two public APT attacks respectively. As S-HOLMES follows the original HOLMES design, threat and benign scores do not change when there is no adversarial activities per the corresponding attack stage.

As evident from S-HOLMES results, the score deviation between threat and benign scores starts to clearly emerge from stage six, for most of attack scenarios and hence, S-HOLMES can perfectly capture all attack artifacts from the Complete Mission stage. On the contrary, S-HOLMES is unable to detect any of the attack artifacts for the first three attack stages (Initial Compromise, Establish Foothold, and Escalate Privileges), for most of the attack scenarios as the threat scores do not reach the threshold value. Starting from stage 4 (Internal Reconnaissance), S-HOLMES starts to generate alerts with a high rate of false positives as the threat and benign scores are very close.

Scenario No.	Measurements	S-HOLMES						
		S1	S2	S3	S4	S5	S6	S7
1	Threat Score	13	108	108	1987	1987	55342	1163881
	Benign Score	13	108	108	1328	1328	1328	1328
2	Threat Score	13	108	108	1988	1988	55379	55379
	Benign Score	13	108	108	1331	1331	1331	1331
3	Threat Score	13	108	108	1987	1987	55342	1163881
	Benign Score	13	29	298	298	298	298	298
4	Threat Score	2	5	74	903	903	25153	25153
	Benign Score	7	62	638	638	638	638	638
5	Threat Score	13	29	432	7946	7946	221381	4648997
	Benign Score	7	60	897	16504	16504	16504	16504
6	Threat Score	7	17	247	4530	4530	126199	2653973
	Benign Score	7	60	897	16504	16504	16504	16504
7	Threat Score	13	29	297	5466	96092	96092	96092
	Benign Score	13	108	1110	13650	13650	13650	13650
8	Threat Score	13	108	1110	20414	20414	568743	568743
	Benign Score	13	13	108	1615	19844	19844	19844

Table 2. S-HOLMES evaluation on DARPA attack scenarios. Threat and benign scores are calculated per every attack stage (S1, S2, ..., S7).

APTs	CTI Report	Report Year	Measurements	S-HOLMES						
				S1	S2	S3	S4	S5	S6	S7
APT41	FireEye [25]	2019	Threat Score	2	18	18	226	226	9002	308730
			Benign Score	2	18	18	338	338	338	338
APT35	Darktrace [15],	2021	Threat Score	7	62	62	1133	1133	45089	948268
	FireEye [28]	2019	Benign Score	10	85	85	1039	1039	1039	1039

Table 3. S-HOLMES evaluation on public attacks.

On the other hand, APTHunter precisely captured the attack artifacts as early as in the Initial Compromise stage. In terms of false positives, APTHunter encountered low performance for attack scenario six (for the Initial Compromise stage, precision = 0.46), where the DARPA performers added noise to the dataset by forcing the *Nginx* process to communicate to external IP addresses in a way that mimics reverse shell activities.[10]

In terms of false negatives, APTHunter missed one of the Initial Compromise stage activities for attack scenario 5 as the adversary used stolen credentials to access the victim device, resulting in recall of (0.715).

Once the adversary implanted a web shell on the victim's device, APTHunter could perfectly detect the activities in the next stage (Establish Foothold).

5.2 Evaluation on Other Real-World APT Attacks

We additionally evaluate APTHunter on two other real-world APTs (not included in the DARPA tests), and compare its effectiveness with S-HOLMES; see Table 5. Note that we use the same set of provenance queries as used with DARPA adversarial scenarios to understand how generic our queries are for other APTs. Our provenance queries represent attack behaviors rather than certain IOCs. For a given attack technique, the attack

[10]Review of the attack provenance graph generated by APTHunter also did not show any further attack activities for the next stages.

Scenario No.	Measurements	S-HOLMES							APTHunter						
		S1	S2	S3	S4	S5	S6	S7	S1	S2	S3	S4	S5	S6	S7
1	Precision	0	0	-	0.143	-	1	1	1	1	-	1	-	1	1
	Recall	0	0	-	0.821	-	1	1	0.861	1	-	1	-	1	1
	F-score	0	0	-	0.244	-	1	1	0.925	1	-	1	-	1	1
2	Precision	0	0	-	0.253	-	1	-	1	1	-	1	-	1	-
	Recall	0	0	-	0.921	-	1	-	0.832	1	-	1	-	1	-
	F-score	0	0	-	0.397	-	1	-	0.908	1	-	1	-	1	-
3	Precision	0	0	-	1	-	1	1	1	1	-	1	-	1	1
	Recall	0	0	-	0.846	-	1	1	0.824	1	-	1	-	1	1
	F-score	0	0	-	0.917	-	1	1	0.904	1	-	1	-	1	1
4	Precision	0	0	0	0.215	-	1	-	1	1	1	1	-	1	-
	Recall	0	0	0	0.761	-	1	-	0.8	1	1	1	-	1	-
	F-score	0	0	0	0.43	-	1	-	0.889	1	1	1	-	1	-
5	Precision	0	0	0	0.162	-	1	1	1	1	1	1	-	1	1
	Recall	0	0	0	0.734	-	1	1	0.715	1	1	1	-	1	1
	F-score	0	0	0	0.266	-	1	1	0.834	1	1	1	-	1	1
6	Precision	0	0	0	0.137	-	1	1	0.46	1	1	1	-	1	1
	Recall	0	0	0	0.781	-	1	1	0.867	1	0.882	1	-	1	1
	F-score	0	0	0	0.233	-	1	1	0.601	1	0.937	1	-	1	1
7	Precision	0	0	0	0.491	0.738	1	-	0.957	1	0.99	1	1	1	-
	Recall	0	0	0	0.762	1	1	-	1	1	1	1	1	1	-
	F-score	0	0	0	0.597	0.849	1	-	0.978	1	0.995	1	1	1	-
8	Precision	0	0	1	0.135	0.546	1	-	0.988	1	1	1	1	1	-
	Recall	0	0	0.977	0.592	0.988	1	-	1	1	1	1	1	1	-
	F-score	0	0	0.988	0.219	0.703	1	-	0.944	1	1	1	1	1	-

🟩 Measurement ≥ 0.9 🟨 0.9 > Measurement ≥ 0.6 🟧 0.6 > Measurement ≥ 0.4 🟥 0.4 > Measurement

Table 4. Evaluation on DARPA attack scenarios. The detection results are per every attack stage S1: Initial Compromise, S2: Establish Foothold, S3: Escalate Privileges, S4: Internal Reconnaissance, S5: Lateral Movement, S6: Complete Mission, S7: Cleanup Tracks), and are based on the measured Precision, Recall and F-score. Note: we use '-' when there are no adversarial activities for the corresponding attack stage. We use '0' to indicate that no alerts are generated by the detection system.

behavior often remains constant even with using different attack tools (e.g., RATs). We aim to test the performance of our system against new APT attacks which use similar attack techniques (but with different tools).

Attack Scenarios Description. We simulate two recent APT attack scenarios, covering all their attack stages. Different artifacts from the available CTI reports [15, 25, 28] are collected and mutated, given that it is a common practice among attackers to mutate attack artifacts to evade detection systems. To that end, we use publicly available mutated versions of the malware used in those APTs. We also used another set of IP addresses for C&C. While mutating attack artifacts, we make sure that the attack behavior in every stage remains as is.

Double Dragon APT41. APT41 is attributed to a Chinese cyber threat group who carry out espionage and financially motivated activities, targeting mainly the healthcare, high-tech, telecommunication, and gaming sectors [25]. In our simulation, attackers start by exploiting a TeamViewer buffer overflow vulnerability as an entry point to the victim network and use that to plant a web shell. The web shell is used to transfer Gh0st RAT to Establish Foothold. Gh0st is used to send a reverse shell to the C2 under the attacker's control. Next, Mimikatz is used to dump credentails as part of the Escalate Privileges stage. After that, Gh0st is configured to run on the startup and to send a shell every 60 seconds to maintain persistence. Then, Internal Reconnaissance activities are performed to identify the victim network. Port scanning and service enumeration among other internal network enumeration activities are performed throughout this stage. Technical account credentials are found unprotected in the victim device used to connect to a file server. The adversary exfiltrates classified documents, credentials repositories found in the system, and concludes the attack by clearing the audit log as part of the Cleanup Tracks stage.

Charming Kitten APT35. APT35 is attributed to an Iranian cyber-espionage group which target U.S. and Middle eastern military, government organizations, and media and energy sectors [26]. In our scenario, the victim is lured to visit a malicious website with a vulnerable Firefox browser which is exploited as part of the Initial Compromise stage. Once the adversaries have access to the victim device, they deploy a PHP web shell and used it to deploy PUPYRAT, a cross-platform and multi function RAT, as part of the Establish Foothold stage. SMB scanning activities are performed as part of the Internal Reconnaissance stage to enumerate the victim host operating system, open and hidden shares, and user generic and service accounts. Generic and service accounts [14] are usually privileged accounts and likely not highly secured [42] (possibly even using common passwords). The adversaries are able to compromise credentials of a generic account using password spraying technique [53]. Next, they use the generic account to add root privileges to an account under their control as part of the Escalate Privileges stage. The adversaries, then, exfiltrate data from the archived storage and delete system log files as part of the Complete Mission and the Cleanup Tracks stages respectively.

APTs	CTI Report	Report Year	Measurements	S-HOLMES							APTHunter						
				S1	S2	S3	S4	S5	S6	S7	S1	S2	S3	S4	S5	S6	S7
APT41	FireEye [25]	2019	Precision	0	0	0	0	0	1	1	1	1	1	0.778	1	1	1
			Recall	0	0	0	0	0	0.5	0.6	1	1	1	1	1	1	1
			F-score	0	0	0	0	0	0.667	0.75	1	1	1	0.875	1	1	1
APT35	Darktrace [15], FireEye [28]	2021 2019	Precision	0	0	0	0.286	0	1	1	1	1	1	0.8	1	1	1
			Recall	0	0	0	0.333	0	1	1	1	1	1	1	1	1	1
			F-score	0	0	0	0.308	0	1	1	1	1	1	0.889	1	1	1

🟩 Measurement ≥ 0.9 🟨 0.9 > Measurement ≥ 0.6 🟧 0.6 > Measurement ≥ 0.4 🟥 0.4 > Measurement

Table 5. Evaluation on other real-world APT attacks.

Detection Results. We compare APTHunter with the results of S-HOLMES; see Table 5 (for S-HOLMES' threat and benign scores for APT attack stages, see Table 3).

Similar to DARPA tests, S-HOLMES usually performs well for detecting APT campaigns at later stages when the threat and benign scores start to differentiate. For APT41, S-HOLMES starts to detect attack artifacts from stage 6 (Complete Mission) with no false alarms (perfect precision). However, S-HOLMES was not able to detect removing of classified documents nor clearing the log files (artifacts in APT stages 6 and 7 respectively), resulting in several false negatives (recall = 0.5 and 0.6 for stages 6 and 7 respectively).

For Charming Kitten APT35, S-HOLMES was able to detect all attack artifacts in stages 6 and 7 perfectly without any false negatives nor false positives.

On the contrary, when the goal is to detect APT campaign in early stages, S-HOLMES was not able to detect the attack artifacts in both APT Attack campaigns.

This is because HOLMES is solely based on the abstract definition of TTPs defined by the MITTE ATT&CK framework. Attack behaviors included in the CTI reports are missed. In such situations, APTHunter has a better chance to detect the attacks in early stages even with mutated attack artifacts, as the attack behavior often remains constant among mutations. As shown in results, APTHunter consistently outperforms in all attack stages.

5.3 Evaluation on Benign Datasets

To further evaluate APTHunter's performance in terms of false positives, we use two weeks of benign enterprise activities, generated as part of the adversarial engagements 3 and 5 by the DARPA TC program. Benign activities include regular user activities (e.g., web browsing, software updates), as well as system administration activities (e.g., system configuration, remote connections). We evaluated our system to see if it generates false alerts as per every attack stage and APTHunter generated no false alerts for all attack stages.

5.4 Efficiency

We evaluate the performance of APTHunter in terms of the audit log consumption time, the provenance graph occupied memory and the attack search time to generate alerts.

Audit Logs Consumption and Provenance Graph Generation. Consumption time represents the time needed to consume all the audit log events per every stream from disk for building the provenance graph in memory. This time varies based on the structure of the audit log for different operating systems and the intensity of activities on each stream which is, to some extent, reflected by the log size on disk. For APTHunter, the consumption time includes also the time taken to normalize and generate the canonical and compact form of the audit log, which is used to generate the provenance graph. As a result, consumption time of S-HOLMES is slightly less than consumption time of APTHunter; see Figure 3.

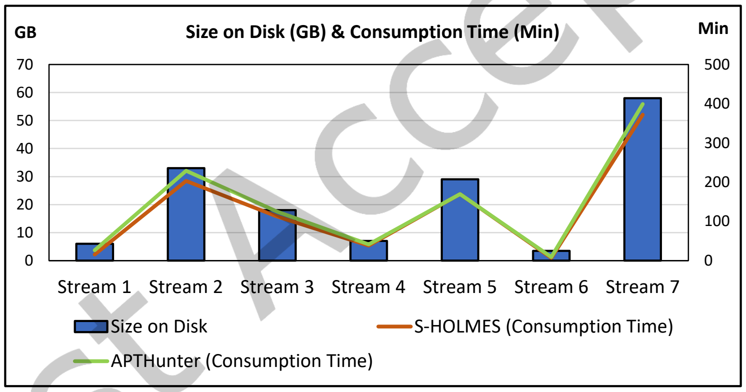

Fig. 3. Log size on disk (in GB) and consumption time (from the moment the log is generated all the way to generating the provenance graph) (in minutes). The columns show the uncompressed log size on disk, while the brown and green lines show the consumption time for S-HOLMES and APTHunter respectively.

Occupied Memory. Colored lines in Figure 6 represent the memory consumption by APTHunter throughout the ingestion of the different streams. The maximum memory consumed by APTHunter is 107.30 MB. This allows APTHunter to cover more than 1200 hosts in one dedicated server with 128 GB of memory. In Figure 4, we show the predicated number of hosts that APTHunter can support in one dedicated server for different memory settings. The memory consumption by the operating system, Kafka stream processor, Neo4j engine, and the other system needed processes are already considered in the prediction. To examine the compression ratio by APTHunter, we disabled the garbage collector process to allow consuming all stream records in memory. By comparing the log size on disk versus the provenance graph consumed memory, we notice that APTHunter has an average compression ratio of around 93% for the provenance graph representation based on our compact and reduced event log form, while the average compression ratio for S-HOLMES is around 75% as shown in Figure 5.

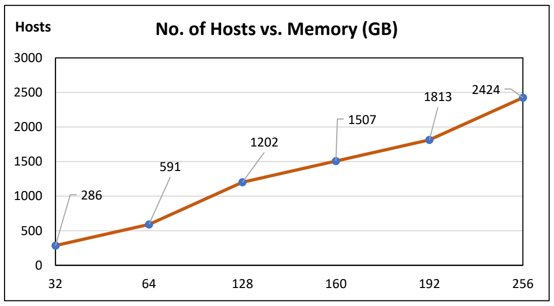

Fig. 4. The number of hosts that can be supported by APTHunter in one APTHunter's dedicated server versus different server memory sizes (in GB). We also consider the time taken by the operating system and the other needed processes (e.g., Neo4j engine).

Search Time. Lines in Figure 5 show the search time for both systems. The search time is measured from the moment the rules (in S-HOLMES), or the Provenance Queries (in APTHunter) are submitted until ending their execution on different streams. Obviously, the search time is directly proportional to the stream size. However, we also notice that nodes with matched attack conditions have a more significant affect on the search time, as all sub nodes need to be investigated and hence, search time increases accordingly. To measure the search time overhead by the garbage collector, we measure the search time when the garbage collector is enabled and one more time when we disable it. As shown in the figure, APTHunter surpasses S-HOLMES in the search time for all streams with average search time of 22% of the time needed by S-HOLMES (when GC is enabled) and 18% (when GC is disabled).

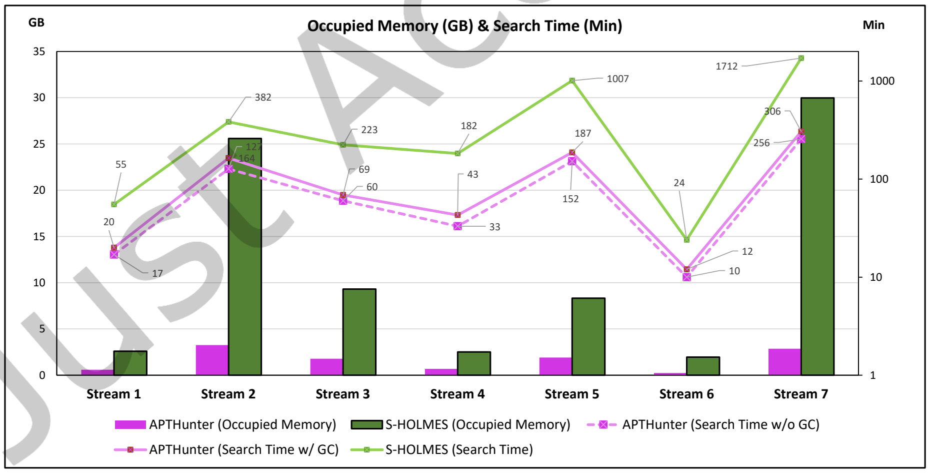

Fig. 5. Provenance graph occupied memory and attack search time for S-HOLMES and APTHunter. The green and purple columns represent the total memory consumption (in GB) by S-HOLMES and APTHunter respectively, while the light-colored lines represent the attack search time (in minutes). The dash line shows search time by APTHunter when the garbage collector (GC) is disabled.

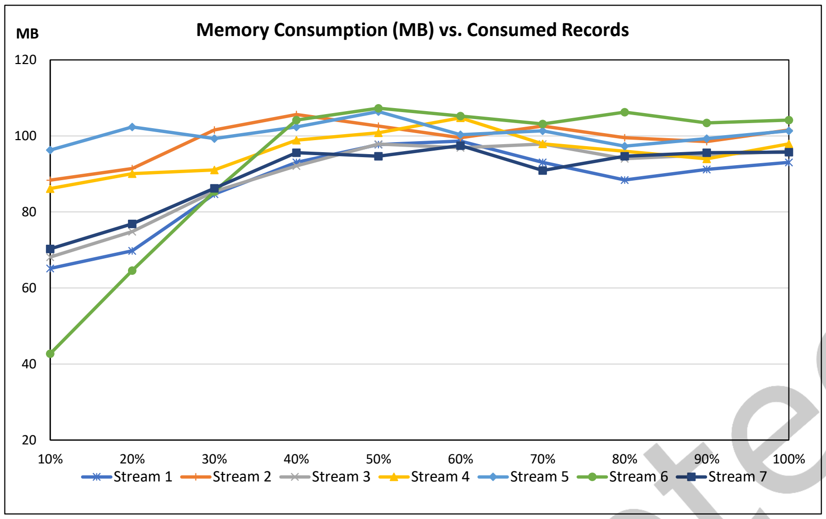

Fig. 6. Provenance graph occupied memory (in MB) for APTHunter.

6 LIMITATIONS

Even though APTHunter is designed to detect the attacker's behavior rather than detecting specific set of IOCs that can be mutated by the attackers, APTHunter still shares limitations with other rule-based detection systems. In particular, APTHunter is based on attack descriptions included in the published CTI reports. If the attacker's behavior changes, APTHunter must be configured with the new attack behaviour to be able to detect it. Also as in rule-based detection systems, detection signals are tuned by threshold values (e.g., IP scan count $\geq$ 10); attack activities that fly under that threshold will not be detected.

In addition, adversaries who get access to the network using stolen credentials or similar means cannot be detected by APTHunter until a malicious behavior occurs. Behavior-based systems work better on this by detecting unusual connections to the network (e.g., a privileged user account was used to access an application server for the first time), albeit with the cost of high volume of false alerts.

Moreover, APTHunter does not trigger alerts based on correlatation between detections from different hosts. We leave this for the Security Orchestration, Automation and Response (SOAR) systems.[11] In SOAR, use cases can be created to trigger alerts in such activities.

7 RELATED WORK

This work lies at the intersection of host-based intrusion detection, provenance-based causal analysis, and malicious behavior discovery. Therefore, we place APTHunter in the context of literature in these areas.

Provenance-Based Detection Systems. HOLMES [51] is the first system to apply event matching techniques to data provenance to detect APT campaigns. They also proposed methods for threat score assignment and noise reduction. However, it assumes 100% event logs retention to calculate threat and benign scores and generate alerts accordingly [34].

[11]Such as CORTEX (https://www.paloaltonetworks.com/cortex/cortex-xsoar) and Helix (https://www.fireeye.com/products/helix/soar.html).

This is practically infeasible in Endpoint Detection and Response (EDR) systems for space constraint, a limitation addressed by APTHunter through the introduction of prerequisite queries. Second, to suppress false alerts, HOLMES assumes having a quantitative normal behavior database of system entities. In enterprise setting, that is an elusive endeavor due to concept drift as benign behavior changes. In contrast, APTHunter mitigates false alerts through the correlation of attack behavior with matches from the prerequisite queries.

POIROT [50] extracts attack artifacts from APT Cyber Threat Intelligence (CTI) reports for a selected set of APT campaigns and modeled them into attack graphs. The authors used their proposed graph alignment algorithm to determine if the attack graph for the APT manifests itself inside the whole system provenance graph as an indication of compromise. However, POIROT assumed that APT attack stages will occur consecutively and attack activities are sequentially linked to each other, failing to detect persistent attacks that can conceal their causally related artifacts by spanning them in a long period of time. Unlike POIROT, APTHunter extracts attack artifacts from CTI reports to formalize provenance queries for granular attack detection.

UNICORN [33] is an anomaly-based system, aims to generate a set of provenance graphs that represent known good behaviors. The authors proposed a clustering-based technique to compare between the known-good provenance graphs and the whole system provenance to detect any anomaly. UNICORN inherits drawbacks of anomaly-based IDS systems including high rate of false alerts, besides the difficulty to have a complete and updated model of benign behaviors.

Alert Correlation Detection Systems. Event correlation is another line of research which involves statistical-based, heuristic-based, and probabilistic-based event correlation techniques [32, 59, 68] to generate alerts. In industry, Security Information and Event Management (SIEM) systems (e.g., Arcsight [6], ELK [21]) leverage similar event correlation techniques for event correlation and alert generation. These techniques use feature-based correlation techniques unrelated to causal analysis between the events. In contrast, APTHunter can establish causal dependencies between events.

Graph Query Processing Systems. Previous work [31, 69, 78] incorporated graph query processing methods and graph indexing and optimization techniques to support timely investigations. AIQL [30] and SAQL [29] took steps further and designed a domain-specific anomaly query processing systems to query specified anomalies on the system event logs, which can help in attack forensics and investigations. CamQuery [62] is a graph query framework that supports realtime analysis on provenance graph while imposing minimal overheads on systems execution. These works are orthogonal to APTHunter and can be leveraged to implement our ProvQuery approach.

8 CONCLUSION

We present APTHunter, an APT detector in early stages. APTHunter formulates the cyber threat hunting as a provenance query matching problem to reliably pinpoint APT artifacts in the system event logs based on the attack behavior in the published CTI reports. We evaluate APTHunter on two real-world APT campaigns and on eight attack scenarios conducted by DARPA professional red-team, over different platforms on enterprise settings, with millions of event log records. APTHunter outperforms with the outstanding provenance graph compression ratio of 93% and prompt detection (with high confidence) of attack artifacts accompanied by an explanation of the attack evolution on the system. Automating the analysis and extraction of intelligence information from CTI reports for the purpose of automatic generation of the provenance queries is an interesting research direction for our future work.

REFERENCES

[1] ABUSE. [n.d.]. Fighting Malware and Botnets. https://abuse.ch/.

[2] Security Affairs. 2022. China-linked APT10 Target Taiwan’s financial trading industry. https://securityaffairs.co/wordpress/128273/apt/apt10-targets-taiwan-financial-trading.html.

[3] Olusola Akinrolabu, Ioannis Agrafiotis, and Arnau Erola. 2018. The challenge of detecting sophisticated attacks: Insights from SOC Analysts. In *ARES*. 1–9.
[4] AlienVault. [n.d.]. AlienVault Open Threat Exchange. https://otx.alienvault.com/browse/global.
[5] Apache. 2017. Kafka Streams. https://kafka.apache.org/documentation/streams/.
[6] ArcSight. 2021. ArcSight Enterprise Security Manager. https://www.microfocus.com/en-us/cyberres/secops/arcsight-esm.
[7] Frederick Barr-Smith, Xabier Ugarte-Pedrero, Mariano Graziano, Riccardo Spolaor, and Ivan Martinovic. 2021. Survivalism: Systematic Analysis of Windows Malware Living-Off-The-Land. In *S&P*.
[8] BeyondTrust. 2020. Cyber-Attack Chain. www.beyondtrust.com/resources/glossary/cyber-attack-chain.
[9] Bricata. 2021. Layers of Cybersecurity: Signature Detection vs. Network Behavioral Analysis. https://bricata.com/blog/signature-detection-vs-network-behavior/.
[10] J Briffaut, P Clemente, JF Lalande, and J Rouzaud-Cornabas. 2011. From Manual Cyber Attacks Forensic to Automatic Characterization of Attackers' Profiles. *Université d'Orléans* (2011).
[11] CISA. 2020. APT Groups Target Healthcare and Essential Services. https://us-cert.cisa.gov/ncas/alerts/AA20126A.
[12] CrowdStrike. [n.d.]. Advanced Persistent Threat Definition. https://www.crowdstrike.com/cybersecurity-101/advanced-persistent-threat-apt/.
[13] CrowdStrike. 2018. Is There Such a Thing as a Malicious PowerShell Command? www.crowdstrike.com/blog/is-there-such-a-thing-as-a-malicious-powershell-command/.
[14] CYBERARC. 2017. 7 Types of Privileged Accounts. www.cyberark.com/resources/blog/7-types-of-privileged-accounts-service-accounts-and-more.
[15] Darktrace. 2021. APT35 'Charming Kitten' discovered in a pre-infected environment. https://www.darktrace.com/en/blog/apt-35-charming-kitten-discovered-in-a-pre-infected-environment/.
[16] DARPA. [n.d.]. Transparent Computing. https://www.darpa.mil/program/transparent-computing.
[17] DARPA. 2018. Transparent Computing, TA5.1 Ground Truth Report Engagement 3. https://drive.google.com/file/d/1mrs4LWkGk-3zA7t7v8zrhm0yEDHe57QU/view?usp=sharing.
[18] DARPA. 2019. Transparent Computing, TA5.1 Final Report Engagement 5. https://drive.google.com/file/d/1cc3C5JW-Kn-VdXqeBGwvHBKSdR_YmSGj/view?usp=sharing.
[19] Defence Research and Development Canada. 2020. TA-35—Cyber Threat Data Model and Use Cases. www.hhs.gov/sites/default/files/apt-and-cybercriminal-targeting-of-hcs.pdf.
[20] EclecticIQ. [n.d.]. Intelligence at the core. https://www.eclecticiq.com/.
[21] Elastic. 2021. SIEM for the modern SOC. https://www.elastic.co/siem.
[22] Finextra. 2021. The state of cybersecurity in financial services. https://www.finextra.com/blogposting/20387/the-state-of-cybersecurity-in-financial-services.
[23] FireEye. [n.d.]. Redline. https://www.fireeye.com/services/freeware/redline.html/.
[24] FireEye. [n.d.]. Threat Intelligence Reports. https://www.fireeye.com/current-threats/threat-intelligence-reports.html.
[25] FireEye. 2019. Special Report: Double Dragon APT41, a dual espionage and cyber crime operation. https://content.fireeye.com/apt-41/rpt-apt41.
[26] FireEye. 2021. Cyber-Attack Chain. https://www.fireeye.com/current-threats/apt-groups.html.
[27] FireEye. 2021. Threat Intelligence Reports by Industry. https://www.fireeye.com/current-threats/reports-by-industry.html.
[28] FireEye-Mandiant. 2018. M-Trends 2018 report. https://www.fireeye.com/content/dam/collateral/en/mtrends-2018.pdf.
[29] Peng Gao, Xusheng Xiao, Ding Li, Zhichun Li, Kangkook Jee, Zhenyu Wu, Chung Hwan Kim, Sanjeev R Kulkarni, and Prateek Mittal. 2018. SAQL: A stream-based query system for real-time abnormal system behavior detection. In *USENIX*. 639–656.
[30] Peng Gao, Xusheng Xiao, Zhichun Li, Fengyuan Xu, Sanjeev R Kulkarni, and Prateek Mittal. 2018. AIQL: Enabling efficient attack investigation from system monitoring data. In *USENIX*. 113–126.
[31] Rosalba Giugno and Dennis Shasha. 2002. Graphgrep: A fast and universal method for querying graphs. In *Pattern Recognition*, Vol. 2. IEEE, 112–115.
[32] Guofei Gu, Phillip A Porras, Vinod Yegneswaran, Martin W Fong, and Wenke Lee. 2007. Bothunter: Detecting malware infection through ids-driven dialog correlation.. In *USENIX Symposium*, Vol. 7. 1–16.
[33] Xueyuan Han, Thomas Pasquier, Adam Bates, James Mickens, and Margo Seltzer. 2020. Unicorn: Runtime provenance-based detector for advanced persistent threats. In *NDSS*.
[34] Wajih Ul Hassan, Adam Bates, and Daniel Marino. 2020. Tactical provenance analysis for endpoint detection and response systems. In *S&P*. IEEE, 1172–1189.
[35] Md Nahid Hossain, Sadegh M Milajerdi, Junao Wang, Birhanu Eshete, Rigel Gjomemo, R Sekar, Scott Stoller, and VN Venkatakrishnan. 2017. SLEUTH: Real-time attack scenario reconstruction from COTS audit data. In *USENIX*. 487–504.
[36] Ghaith Husari, Ehab Al-Shaer, Mohiuddin Ahmed, Bill Chu, and Xi Niu. 2017. Ttpdrill: Automatic and accurate extraction of threat actions from unstructured text of cti sources. In *ACSAC*. 103–115.

[37] Imperva. [n.d.]. Data Loss Prevention (DLP). https://www.imperva.com/learn/data-security/data-loss-prevention-dlp.

[38] Kaspersky. [n.d.]. BlackEnergy APT Attacks in Ukraine. https://www.kaspersky.com/resource-center/threats/blackenergy.

[39] Kaspersky. 2022. APT trends report Q1 2022. https://go.kaspersky.com/rs/802-IJN-240/images/Kaspersky_APT_trends_Q1_2022.pdf.

[40] Samuel T King and Peter M Chen. 2005. Backtracking intrusions. *ACM Transactions on Computer Systems* 23, 1 (2005), 51–76.

[41] Samuel T King, Zhuoqing Morley Mao, Dominic G Lucchetti, and Peter M Chen. 2005. Enriching Intrusion Alerts Through Multi-Host Causality.. In *NDSS*.

[42] KPMG. [n.d.]. The hidden security risks from service accounts. https://advisory.kpmg.us/articles/2021/hidden-security-risks-service-accounts.html.

[43] Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. 2013. High Accuracy Attack Provenance via Binary-based Execution Partition.. In *NDSS*.

[44] Shiqing Ma, Xiangyu Zhang, and Dongyan Xu. 2016. Protracer: Towards Practical Provenance Tracing by Alternating Between Logging and Tainting.. In *NDSS*.

[45] MITRE Matrix. [n.d.]. http://https://attack.mitre.org/..

[46] McAfee. [n.d.]. McAfee Advanced Threat Defense. https://www.mcafee.com/enterprise/en-ca/products/advanced-threat-defense.html.

[47] McAfee. [n.d.]. What Is the MITRE ATT&CK. https://www.mcafee.com/enterprise/en-ca/security-awareness/cybersecurity/what-is-mitre-attack-framework.html.

[48] Derrick McKee, Yianni Giannaris, Carolina Ortega Perez, Howard Shrobe, Mathias Payer, Hamed Okhravi, and Nathan Burow. 2022. Preventing Kernel Hacks with HAKC. In *Proceedings 2022 Network and Distributed System Security Symposium. NDSS*, Vol. 22. 1–17.

[49] Noor Michael, Jaron Mink, Jason Liu, Sneha Gaur, Wajih Ul Hassan, and Adam Bates. 2020. On the forensic validity of approximated audit logs. In *Annual Computer Security Applications Conference*. 189–202.

[50] Sadegh M Milajerdi, Birhanu Eshete, Rigel Gjomemo, and VN Venkatakrishnan. 2019. Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting. In *CCS*.

[51] Sadegh M Milajerdi, Rigel Gjomemo, Birhanu Eshete, Ramachandran Sekar, and VN Venkatakrishnan. 2019. HOLMES: Real-time APT detection through correlation of suspicious information flows. In *S&P*. IEEE, 1137–1152.

[52] Simon Miles, Paul Groth, Miguel Branco, and Luc Moreau. 2007. The requirements of using provenance in e-science experiments. *Journal of Grid Computing* 5, 1 (2007), 1–25.

[53] MITRE. [n.d.]. MITRE Matrix, Brute Force: Password Spraying. https://attack.mitre.org/techniques/T1110/003/.

[54] MITRE. [n.d.]. MITRE Matrix, Exploit Public-Facing Application. https://attack.mitre.org/techniques/T1190/.

[55] MITRE. [n.d.]. MITRE Matrix, OS Credential Dumping: LSASS Memory. https://attack.mitre.org/techniques/T1003/001.

[56] N-ABLE. 2021. Intrusion Detection System (IDS): Signature vs. Anomaly-Based. https://www.n-able.com/blog/intrusion-detection-system/.

[57] Neo4j. [n.d.]. The Native Graph Database for Today's Connected Applications. https://neo4j.com/product/neo4j-graph-database/.

[58] Nextron Systems. 2021. LOKI, Open-Source IOC Scanner. https://www.nextron-systems.com/loki/.

[59] Peng Ning and Dingbang Xu. 2003. Learning attack strategies from intrusion alerts. In *CCS*.

[60] U.S. Department of Health and Human Services (HHS). 2022. Health Sector Cybersecurity: 2021 Retrospective and 2022 Look Ahead. https://www.hhs.gov/sites/default/files/2021-retrospective-and-2022-look-ahead-tlpwhite.pdf.

[61] OWASP. 2018. Kerberoasting. https://owasp.org/www-pdf-archive/OWASP_Frankfurt_-44_Kerberoasting.pdf.

[62] Thomas Pasquier, Xueyuan Han, Thomas Moyer, Adam Bates, Olivier Hermant, David Eyers, Jean Bacon, and Margo Seltzer. 2018. Runtime analysis of whole-system provenance. In *CCS*.

[63] Kexin Pei, Zhongshu Gu, Brendan Saltaformaggio, Shiqing Ma, Fei Wang, Zhiwei Zhang, Luo Si, Xiangyu Zhang, and Dongyan Xu. 2016. Hercule: Attack story reconstruction via community discovery on correlated log graph. In *ACSAC*. 583–595.

[64] Red Canary. [n.d.]. Red Canary 2021 Threat Detection Report. https://redcanary.com/threat-detection-report/.

[65] Secureworks. [n.d.]. Advanced Persistent Threats: Learn the ABCs of APTs. https://www.secureworks.com/blog/advanced-persistent-threats-apt-a.

[66] Panda Security. 2019. How Endpoint Detection and Response gave rise to Threat Hunting. https://www.pandasecurity.com/en/mediacenter/security/edr-threat-hunting.

[67] Financial Services Information Sharing and Analysis Center (FS-ISAC). 2022. Navigating Cyber. https://www.fsisac.com/hubfs/NavigatingCyber-2022/NavigatingCyber2022-TLPWHITE-FIN.pdf.

[68] Yun Shen and Gianluca Stringhini. 2019. Attack2vec: Leveraging temporal word embeddings to understand the evolution of cyberattacks. In *USENIX*. 905–921.

[69] Zhao Sun, Hongzhi Wang, Haixun Wang, Bin Shao, and Jianzhong Li. 2012. Efficient subgraph matching on billion node graphs. In *VLDB*.

[70] Symantec. 2019. Buckeye: Espionage Outfit Used Equation Group Tools Prior to Shadow Brokers Leak. https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence/buckeye-windows-zero-day-exploit.

[71] The New York Times. 2019. How Chinese Spies Got the N.S.A.'s Hacking Tools, and Used Them for Attacks. https://www.nytimes.com/2019/05/06/us/politics/china-hacking-cyber.html.

[72] The U.S. Department of Health and Human Services. 2020. APT and Cybercriminal Targeting of HCS. https://www.hhs.gov/sites/default/files/apt-and-cybercriminal-targeting-of-hcs.pdf.

[73] Jacob Torrey. 2020. Transparent Computing Engagement 5 Data Release. https://github.com/darpa-i2o/Transparent-Computing.

[74] Varonis. [n.d.]. Threat detection & response. https://www.varonis.com/solutions/threat-detection-response/.

[75] Varonis. 2020. What is an Advanced Persistent Threat? www.varonis.com/blog/advanced-persistent-threat/.

[76] Vectra. [n.d.]. Network detection and response built on artificial intelligence. https://www.vectra.ai/products/cognito-platform.

[77] Qi Wang, Wajih Ul Hassan, Ding Li, Kangkook Jee, Xiao Yu, Kexuan Zou, Junghwan Rhee, Zhengzhang Chen, Wei Cheng, Carl A Gunter, et al. 2020. You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis.. In *NDSS*.

[78] Xiaoli Wang, Xiaofeng Ding, Anthony KH Tung, Shanshan Ying, and Hai Jin. 2012. An efficient graph indexing method. In *Data Engineering*. IEEE, 210–221.

[79] Zhang Xu, Zhenyu Wu, Zhichun Li, Kangkook Jee, Junghwan Rhee, Xusheng Xiao, Fengyuan Xu, Haining Wang, and Guofei Jiang. 2016. High fidelity data reduction for big data security dependency analyses. In *CCS*.

A NORMALIZING LOGS TO CANONICAL FORM

The audit log contains an immense number of raw low level system events that do not readily reveal the causal relationship between system entities. Hence, it cannot be coupled with the high level artifacts in CTI reports for the reasons discussed below. We also discuss our workarounds for generating the normalized event log form.

A.1 Asymmetric System Call Arguments

Every system call (syscall) has its kind of arguments, which are not symmetrically structured across the audit log. Listing 2 shows examples of system call records from the event log for Ubuntu 14.04 (64 bit). For instance, Listing 2(a) shows an event log record for `execve` syscall ($syscall = 59$) with the syscall arguments are distributed in `a0` (/bin/bash), `a1` (/usr/share/gh0st) and `a2` fields. On the other hand, it is clear that `fork` syscall ($syscall = 56$) allocates its main argument, the process ID (PID) of the new forked sub-process, in the `exit` field (see Listing 2(b)). The `chown` syscall family attributes the owner (user) id and group id in fields `a2` and `a3`, respectively. We can see in Listing 2(g) that the owner id and group id are both 0, which refers that the owner is now root. We need to fix this to represent system entities and the relationship between them in symmetric structure to map them to the high level descriptions in the CTI reports.

A.2 System Entities Unique Identifiers

We also observe that no explicit unique identifier is provided for the system entities in the event log record. Even if the PID can uniquely identify a running process at the runtime, the same PID can be later reused by another process after the earlier process terminates. Hence, the PID or the PPID cannot be used to uniquly identify a process and its parent across the system runtime.

In addition, for particular syscalls, such as `execve`, the executed process inherits the PID of the origin (source) process. So technically, the origin process is not clearly identified in the `execve` syscall event. We run a backward tracing [49] on all previously run processes to pinpoint the recent process with the same PID before the new process is started.

A.3 System Call Dependencies

For specific syscalls, event records are dependent on previous calls (e.g., `read` and `write` depend on `open`). As in Listings 2(d) and 2(e), the process `scp` reads and writes an object which is not included in the corresponding log record. A backward scan for the event logs shows that the `open` syscall record (see Listing 2(f)) contains the object to be read or written to, with its identifier (inode). We complement `read` and `write` syscall records with that missing information in our normalized records.

A.4 Data Format/Encoding

The audit log presents syscall arguments in different formats as evident in Listing 2, including: decimal values for `pid, ppid` in all syscalls, hex values for `a1, a2, a3` in `chmod` (see Listing 2(h)), and string values for `exe`. Besides, the same attribute can be in different format based on the syscall; for example `a2, a3` are represented in decimal values for the `chown` syscall family, but in hex for `chmod`. Once the data format representation is determined, it is important to understand the encoding used with every attribute before being able to reveal its value. As an example, `a2, a3` for the `chmod` syscall family are represented in hex and need to be converted into octal format to reveal the permissions granted for the object. Other attributes in the event logs contain encoded packed values. For example, in `connect` (see Listing 2(c)) and `accept` syscalls, the `saddr` attribute which includes the socket information (IP Address and Port) is encoded in hex packed format which needs to be processed to unpack it and retrieve the corresponding IP Address and Port.

To overcome the above challenges, we provide a canonical representation of the audit log with the following features:

- Compact format: we combine records that have the same syscall ID (indicating that they belong to the same operation), in one record in the compact form. For example, `execve` spwans at least the following record types in the audit log: `SYSCALL`, `EXECVE`, `CMD`, `PATH`, and `PROCTITLE`. We correlate all these events and include all the important information in one compact record.
- Unique identifier representation: we create a universal unique identifier (UUID) for every entity (e.g., processes, files, and sockets) in the system to uniquely identify the entity across the runtime. This requires analysis of different syscalls and their arguments and attributes.
- Symmetrical structure: we symmetrically structure every record in the new canonical form across different syscalls. The new record, in our proposal, contains 9 fields which capture the needed information for preserving causality among system entities during the system runtime: Timestamp, ID, SubjectUUID, SubjectProcess, Action, ObjectUUID, ObjectName, ActionDetails, and Hostname.

Our devised canonical log representation provides all the important information needed in attack forensics and real-time attack artifacts detection.

For further reducing the log size while preserving the causality relationship between system entities, we applied a causality preserved reduction technique as in [79]. The core idea is to merge excessive events between the same pair of entities. Every type of event between a pair of entities is maintained in a stack. Every time the same type of event is occurred between the same pair of entities, the event is checked if it can be aggregated with the events in the stack. The aggregation is done if the two events (the new occurred event and the event in the stack) have the same backward and forward trackability. Any two events between two nodes are said to have same backward trakability if no other incoming events to the first node has occured between the end time of the two events in question.

On the other hand, any two events between two nodes are said to have same forward trakability if no other outgoing events from the second node has occured between the start time of the two events in question. If both conditions match, the two events are said to have the same causal dependency and are then merged together. This helps in reducing the intense bursts of semantically similar events which are produced by system daemons and other several applications [79]. This is the last step of log processing done by the LogCore engine before building the whole system provenance graph. A significant point about our provenance graph is that because it is a highly compact version of the audit log, it requires less memory which facilitates real-time ingestion of events and generation of the graph over a long period of time. On this provenance graph, we apply the generated attack behavior queries to pinpoint attack behaviors as in the CTI reports.

B ATTACK-CONDITIONS

The provenance queries are constructed from two queries (prerequisite query and the main query). Here, we are consolidating the conditions for these two building blocks queries for different attack behaviors.

Initial Compromise.
Domain Hijaking (T1584-001).
The prerequisite query:

$$N_1 \in \{BrowserProcesses\} \ \wedge \ R_I = fork \ \wedge N_2 \in \{RemoteAccessProcesses\}$$

The main query:

$$N_3 \in \{N_2\} \ \wedge R_M = connect \ \wedge N_4.ip \notin \{TrustedIPAddresses\}$$

where *RemoteAccessProcesses* is the list of processes used for remote access (e.g., SSHD).
Exploit Public-Facing Applications (T1190).
The prerequisite query:

$$N_1 \in \{PublicFacingProcesses\} \ \wedge R_I = accept \ \wedge N_2.ip \notin \{TrustedIPAddresses\}$$

The main query:

$$N_3 \in \{PublicFacingProcesses\} \ \wedge \ R_M = connect \ \wedge$$
$$N_4.ip \notin \{TrustedIPAddresses\} \ \wedge N_4 \notin \{N_2\}$$

where N_2 and N_4 represent sockets (IP Address and Port).

Non Standard Port (T1571).
The prerequisite query: None
The main query:

$$N_4.ip \notin \{TrustedIPAddresses\} \ \wedge$$
$$Pair(Process\ N_3\ and\ Port\ N_4.port) \notin \{ServicePortList\} \ \wedge \ R_M = connect$$

Establish Foothold.
The prerequisite query:

$$N_1 \in \{CompromisedProcesses\} \ \wedge R_I \in \{fork, execute\} \ \wedge length \le SelectedLength$$

CompromisedProcesses is a set of all processes tagged as compromised from the Initial Compromise stage.
The main query:

$$N_3 \in \{N_2\} \ \wedge \ R_M \in \{fork, execute\} \ \wedge N_4 \in \{CommandLineUtilities\}$$

Escalate Privileges.
Super User Privilege.
The prerequisite query:

$$N_1 \in \{CompromisedProcesses\} \ \wedge R_I \in \{fork, execute\} \ \wedge length \le SelectedLength$$

The main query:

$$N_3 \in \{N_2\} \ \wedge \ R_M \in \{ChangePrincipal\} \ \wedge N_4.uid \in \{SuperUsers\}$$

ChangePrincipal is the set of syscalls that change the owner user. Examples of those syscalls are `chown`, `fchown`, and `lchown` syscalls. $N_4.uid$ is the real user ID of the affected process (N_4).

Super User Utilities.
The prerequisite query:

$$N_1 \in \{CompromisedProcesses\} \ \wedge R_I \in \{fork, execute\} \ \wedge length \leq SelectedLength$$

The main query:

$$N_3 \in \{N_2\} \ \wedge \ R_M \in \{fork, execute\} \ \wedge N_4 \in \{SuperUserUtilities\}$$

Scheduled Tasks.
The prerequisite query:

$$N_1 \in \{cron.d\} \ \wedge R_I \in \{fork, execute\} \ \wedge length \leq SelectedLength$$

The main query:

$$N_3 \in \{N_2\} \ \wedge \ R_M \in \{chown\} \ \wedge N_4.uid \in \{SuperUsers\}$$

Credential Dump.
The prerequisite query:

$$\begin{aligned} R_I \in \{fork, execute\} \wedge \\ (N_1 \in \{CompromisedProcesses\} \ \vee \ N_1 \in \{SuperUserPrivilege\})\wedge \\ length \leq SelectedLength \end{aligned}$$

SuperUserPrivilege is a set of all processes tagged as compromised from *Super User Privilege* technique in Escalate Privileges stage.
The main query:

$$\begin{gathered} N_3 \in \{N_2\} \ \wedge \ N_4.path \ contains \text{ ``}procdump\text{''} \wedge \\ (N_3.euid \in \{SuperUsers\} \ \vee N_3.uid \in \{SuperUsers\}) \ \wedge R_M = execute \end{gathered}$$

$N_3.uid$ and $N_3.euid$ are the real and effective user IDs for the process N_3 respectively.

Valid Domain Accounts.
The prerequisite query:

$$\begin{aligned} &N_1.uid \notin \{DomainUsers\} \ \wedge N_2 \in \{ScriptingProcesses\} \wedge \\ &R_I \in \{fork, execute\} \ \wedge length \leq SelectedLength \end{aligned}$$

The main query:

$$\begin{aligned} &N_3 \in \{N_2\} \ \wedge \ R_M = connect \ \wedge N_3.uid \in \{DomainUsers\} \wedge \\ &N_4 \in \{InternalIPAddresses\} \ \wedge N_4.ip \in \{DomainIPAddresses\} \end{aligned}$$

DomainUsers is the list of users who are authorized to access the domain controller. *ScriptingProcesses* examples include Python and Powershell. *DomainIPAddresses* are IP addresses of the domain controllers. *uid* is the user id for the corresponding process.

Internal Reconnaissance.
Sensitive Access.
The prerequisite query:

$$N_1 \in \{CompromisedProcesses\} \ \wedge R_I \in \{fork, execute\} \ \wedge length \leq SelectedLength$$

The prerequisite query:

$$N_3 \in \{N_2\} \ \wedge \ R_M \in \{open, read\} \ \wedge \ N_4 \in (\{SensitivePaths\} \vee \{SystemCriticalPaths\})$$

Recon Command.
The prerequisite query:

$$N_1 \in \{CompromisedProcesses\} \ \wedge R_I \in \{fork, execute\} \ \wedge length \leq SelectedLength$$

The main query:

$$N_3 \in \{N_2\} \ \wedge \ R_M = execute \ \wedge N_4 \in \{SensitiveCommands\}$$

Port Scan.
The prerequisite query:

$$N_1 \in \{CompromisedProcesses\} \ \wedge R_I \in \{fork, execute\} \ \wedge length \leq SelectedLength$$

The main query:

$$N_3 \in \{N_2\} \ \wedge \ R_M = send \ \wedge N_4 \in \{InternalIPAddresses\} \wedge$$
$$N_4.port \in \{WellKnownPorts\} \ \wedge count(N_4.port) \geq PortCountThres$$

Based on the enterprise settings, the analyst selects the number of ports (*PortCountThres*) at which the provenance query should generate an alert.

Lateral Movement.
The prerequisite query:

$$(N_1 \in \{CompromisedProcesses\} \ \vee N_1 \in \{InternalReconProcesses\}) \wedge$$
$$R_I \in \{fork, execute\} \ \wedge length \leq SelectedLength$$

The main query:

$$N_3 \in \{N_2\} \ \wedge \ R_M = connect \ \wedge N_4 \in \{InternalIPAddresses\}$$

InternalReconProcesses is a set of all processes engaged in Internal Reconnaissance activities.
Complete Mission.
Exfiltration Over C2 Channel.
The prerequisite query:

$$(N_1 \in \{CompromisedProcesses\} \ \vee N_1 \in \{InternalReconProcesses\}) \wedge$$
$$R_I \in \{fork, execute\} \ \wedge length \leq SelectedLength$$

The main query:

$$N_3 \in \{N_2\} \wedge N_4 \notin \{TrustedIPAddresses\} \ \wedge R_M = send$$

send is the family of syscalls that includes syscalls used to send data over the network including sendmsg, sendto, sendfile, etc.
Exfiltration by Bypassing Defense Controls.
The prerequisite query:

$$(N_1 \in \{InternalReconProcesses\} \ \wedge N_1 \in \{EscalatePrivilegeProcesses\}) \wedge \\ R_I \in \{fork, execute\} \wedge length \leq SelectedLength$$

The main query:

$$N_3 \in \{N_2\} \ \wedge N_3.uid \in \{SuperUsers\} \wedge \\ N_4 \notin \{TrustedIPAddresses\} \ \wedge R_M = send$$

EscalatePrivilegeProcesses is a set of all processes with super user privileges detected in Escalate Privileges stage.
Destroy System.
The prerequisite query:

$$(N_1 \in \{CompromisedProcesses\} \ \vee N_1 \in \{InternalReconProcesses\}) \wedge \\ R_I \in \{fork, execute\} \ \wedge length \leq SelectedLength$$

The main query:

$$N_3 \ \in \{N_2\} \ \wedge (N_4 \ \in \{SensitivePaths\} \vee \ \ \in \{SystemFiles\}) \ \wedge R_M \ \ \in \{write, unlink\}$$

SystemFiles can be generic per the operating system and can be based on the CTI report. *SensitivePaths* is customised per every enterprise. Here, enterprises define paths to sensitive files and directories. This includes user drives, internal, confidential, and secret shares. Any suspicious operation on those files will be flagged and an alert will be generated.

Cleanup Tracks.
File Deletion.
The prerequisite query:

$$(N_1 \in \{CompromisedProcesses\} \ \vee N_1 \in \{EscalatePrivilegeProcesses\}) \wedge \\ R_I \in \{fork, execute\} \ \wedge length \leq SelectedLength$$

The main query:

$$N_3 \ \in \ \{N_2\} \ \wedge \ N_4 \ \notin \ \{LogFilesPaths\} \ \wedge \ R_M = unlink$$

Remove Log files.
The prerequisite query:

$$(N_1 \in \{CompromisedProcesses\} \ \vee N_1 \in \{EscalatePrivilegeProcesses\}) \wedge \\ R_I \in \{fork, execute\} \ \wedge length \leq SelectedLength$$

The main query:

$$N_3 \in \{N_2\} \wedge N_4 \in \{LogFilesPaths\} \wedge R_M = unlink$$

Clear Log commands.
The prerequisite query:

$$(N_1 \in \{CompromisedProcesses\} \vee N_1 \in \{EscalatePrivilegeProcesses\}) \wedge \\ R_I \in \{fork, execute\} \wedge length \leq SelectedLength$$

The main query:

$$N_3 \in \{N_2\} \wedge R_M = write \wedge N_4 \in \{LogFilesPaths\}$$

Listing 2 The audit log - system call record examples.

(a) Execve Syscall
```
type=SYSCALL msg=audit(1618071287.244:687514):syscall
=59 success=yes exit=0 a0=1579088 a1=1532a88
a2=163c008 a3=598 ppid=2077 pid=2083 exe="/bin/bash"
type=EXECVE msg=audit(1618071287.244:687514):
a0="/bin/bash" a1="/usr/share/gh0st" a2="start"
type=PATH msg=audit(1618071287.244:687514): item=0
name="/usr/share/gh0st" inode=792704
```

(b) Fork Syscall
```
type=SYSCALL msg=audit(1618072037.116:827848):syscall
=56 success=yes exit=7177 a0=1200011 a1=0 a2=0
a3=7fbec78bd9d0 ppid=7162 pid=7176 exe="/usr/bin/scp"
```

(c) Connect Syscall
```
type=SYSCALL msg=audit(1618071198.080:665224):
syscall=42 success=yes exit=0 a0=aa
a1=7fe95d5b17b0 a2=10 a3=0 items=0 ppid=1407
pid=3461 exe="/usr/lib/firefox/firefox"
type=SOCKADDR msg=audit(1618071198.080:665224):
saddr=020000000DE1BD3D0000000000000000
```

(d) Read Syscall
```
type=SYSCALL msg=audit(1618072049.092:828551):
syscall=0 success=yes exit=221 ppid=7162 pid=7176
exe="/usr/bin/scp"
```

(e) Write Syscall
```
type=SYSCALL msg=audit(1618072049.092:828550):
syscall=1 success=yes exit=80 a0=1 a1=7fff327bd760
a2=50 a3=1 items=0 ppid=7162 pid=7176
exe="/usr/bin/scp"
```

(f) Open Syscall
```
type=SYSCALL msg=audit(1618072049.092:828543):
syscall=2 success=yes exit=3 a1=800 a2=0 a3=8
items=1 ppid=7162 pid=7176 exe="/usr/bin/scp"
type=CWD msg=audit(1618072049.092:828543):
cwd="/home/ubuntu"
type=PATH msg=audit(1618072049.092:828543):
item=0 name="/etc/hosts" inode=2359455
```

(g) Fchownat Syscall
```
type=SYSCALL msg=audit(1617660032.907:154792795):
syscall=260 success=yes exit=0 a0=ffffff9c
a1=1da5cb0 ppid=114171 pid=114172 exe="/bin/chown"
type=PATH msg=audit(1617660032.907:154792795):
name="priv_key.txt" inode=2495043
```

(h) Fchmodat Syscall
```
type=SYSCALL msg=audit(1617660053.219:154794519):
syscall=268 success=yes exit=0 a0=ffffffffffffff9c
a1=11570f0 a2=1ff a3=3c0 ppid=114176 pid=114177
exe="/bin/chmod"
type=CWD msg=audit(1617660053.219:154794519):
cwd="/home/ubuntu"
type=PATH msg=audit(1617660053.219:154794519):
name="gh0st.sh" inode=2495042
```