Threat Detection using Information Flow Analysis on Kernel Audit Logs

by

Sadegh Momeni Milajerdi
M.Sc., Sharif University of Technology, 2012

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2020

Chicago, Illinois

Dedicated to my beloved family

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest gratitude to my adviser, Professor Venkat Venkatakrishnan, for his invaluable guidance, generous support, and constant encouragement throughout my Ph.D. studies. Under his guidance, I learned how to approach a research problem, critically think, analyze, and design practical solutions beneficial to individuals and society. Also, I am deeply grateful to Rigel Gjomemo and Birhanu Eshete for their help and insight in every step of my research.

In addition, I would like to sincerely thank the committee members of my dissertation. I owe a debt of gratitude to Rigel Gjomemo, Jason Polakis, Jon A. Solworth, and R. Sekar, for their generous time and help. The work proposed in this dissertation is implemented on top of a code base initially designed and developed by Professor R. Sekar, and I am deeply grateful to him for the generous guidance and support.

During my Ph.D. studies, I had the chance to collaborate with outstanding researchers out of the University of Illinois at Chicago. I would like to acknowledge Md. Nahid Hossain, Emaad Manzoor, Leman Akoglu, and Scott D. Stoller, who helped and taught me in various ways throughout my Ph.D. I am also deeply appreciative to my internship mentors in Microsoft, Mariusz Jakubowski and Jugal Parikh, who shared their top-notch experience with me. I feel very fortunate that I had a chance to get to know them.

Last but not least, this work would not have been the same without the influence of my family, to whom I am forever indebted. I am always grateful to my father who encouraged me

ACKNOWLEDGMENTS (Continued)

to study abroad, my mother for her patience and endurance, my brother for always being there for me, and my wife for her unconditional love.

SM

CONTRIBUTIONS OF AUTHORS

This dissertation includes material which is previously published, and the copyright statement of these publications indicating our permission for reproducing this material could be found in Appendix D. Here, we describe the inclusion of these published works per each chapter, in addition to the contributions of authors in each work:

- **Chapter 1** introduces the problem we are tackling in this dissertation, overview, and structure of this work.
- **Chapter 2** provides the necessary background on the concepts that will be used throughout the rest of the dissertation. This chapter partially includes excerpts and figures from material that is published in [1–4]. I was the primary author and contributor in [4] and also did the implementation and evaluation of the work. Birhanu Eshete and Rigel Gjomemo generously contributed to the ideas and writing of this paper, and V.N. Venkatakrishnan was the overall supervisor of the work. My contribution in [1–3] is explained below as they comprise the main content of the following chapters.
- **Chapter 3** presents portions of [1], which was a collaborative work between Stony Brook University (SBU) and the University of Illinois at Chicago (UIC). In this work, I was the primary student author from UIC and contributed to significant parts of the implementation and evaluation. Md. Nahid Hossain, Junao Wang, Birhanu Eshete, Rigel Gjomemo, and Scott Stoller contributed to the implementation and writing of the paper.

R. Sekar and V.N. Venkatakrishnan were the supervisors of the wok from SBU and UIC, respectively

- **Chapter 4** covers the work published in [2], of which I was the primary author and also did the implementation and evaluation. Rigel Gjomemo, Birhanu Eshete, and R. Sekar generously contributed to the ideas and writing of the paper, and V.N. Venkatakrishnan was the overall supervisor of the work.
- **Chapter 5** covers the work published in [3], of which I was the primary author and also did the implementation and evaluation. Birhanu Eshete and Rigel Gjomemo generously contributed to the ideas and writing of the paper, and V.N. Venkatakrishnan was the overall supervisor of the work.
- **Chapter 6** covers the related work and partially borrows some content from [1–3] as well.
- **Chapter 7** provides the conclusion along with some suggestions on future research directions.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

APT	Advanced Persistent Threat
ATT&CK	Adversarial Tactics, Techniques, and Common Knowledge
C&C	Command and Control
CPU	Central Processing Unit
CTI	Cyber threat intelligence
CVSS	Common Vulnerability Scoring System
DARPA	Defense Advanced Research Projects Agency
ETW	Event Tracing for Windows
GB	Gigabyte
GPM	Graph Pattern Matching
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HSG	High-level Scenario Graphs
IDS	Intrusion Detection System
IOC	Indicator of compromise
IP	Internet Protocol
IPS	Intrusion Prevention System

LIST OF ABBREVIATIONS (Continued)

JS	JavaScript
LoC	Lines of Code
OS	Operating System
PLC	Programmable Logic Controller
RAT	Remote Administration Tool
SIEM	Security Information and Event Management
TC	Transparent Computing Project (by DARPA)
TTP	Tactics, Techniques, and Procedures
URL	Uniform Resource Locator
WWW	World Wide Web

SUMMARY

Kernel audit logs are a rich source of information containing the history of causal dependencies and information flows among system entities in a host system. The mainstream use of kernel audit logs is for forensic tasks to investigate cyberattacks retrospectively. In this dissertation, we develop efficient methods that make use of kernel audit logs for complex real-time security tasks, such as Advanced and Persistent Threat (APT) detection, attack scenario reconstruction, and cyber threat-hunting. To this end, we first process kernel audit logs into a platform-neutral provenance graph stored in the main memory and use it as a foundation to run various analytics. For APT detection, we develop techniques to produce a detection signal indicating the presence of a coordinated set of suspicious activities. For real-time attack scenario reconstruction, we develop an approach that utilizes information flow policies to identify entities and events that are involved in cyberattacks. For cyber threat-hunting, we develop an inexact graph pattern matching approach to align a query graph extracted from cyber threat intelligence to a provenance graph constructed out of kernel audit logs.

The efficacy of the proposed methods is evaluated against real-world APT scenarios designed for adversarial engagements. These experiments contain millions of records and collectively involve months of audit log collection activity from a variety of hosts that run OS platforms such as Linux, FreeBSD, and Windows. The results indicate that the proposed methods are capable of efficiently searching these audit logs and pinpoint threats in real-time with high precision and low false alarm rate. Besides, these methods effectively produce summaries of attack campaigns that assist investigators in cyber response operations. In summary, this dissertation demonstrates that the low-level causal information inferred from kernel audit logs could be utilized to achieve robust and reliable threat detection methods that efficiently pinpoint threats and reveal the high-level picture of attacks by producing compact visual graphs of attack steps.

CHAPTER 1

INTRODUCTION

We are witnessing a rapid escalation in targeted cyberattacks ("Enterprise Advanced and Persistent Threats (APTs)") [5] conducted by skilled adversaries. These attacks are constantly evolving and getting more sophisticated. To confront these attacks, cyber defenders utilize various solutions such as Security Information and Event Management (SIEM) systems. These defense solutions make reasoning by aggregating security-related events from multiple sources, such as end-user devices, servers, network equipment IDS/IPS, firewalls, and so on. While these systems are generally useful, they often lack (a) an understanding of the complex relationships that exist between alerts and actual intrusion instances and (b) the precision needed to piece together attack steps that take place on different hosts over long periods of time (weeks, or in some cases, months). Instead, significant manual effort and expertise are needed to piece together numerous alarms emitted by multiple security tools. Consequently, many attack campaigns are missed for weeks or even months [6, 7].

The problem of piecing together the causal chain of events leading to an attack was first explored in Backtracker [8, 9] by introducing the notion of provenance graphs constructed from kernel audit logs. In these graphs, vertices represent subjects (processes) and objects (files, sockets), and edges denote audit events (e.g., operations such as read, write, execute, and connect). Provenance graphs provide a detailed history of the causal dependencies and flow of information among system entities. For many years, security analysts have used provenance graphs for forensic tasks to investigate cyberattacks retrospectively.

In this dissertation, we first process kernel audit logs into a provenance graph stored in the main memory, which is suitable for running high-performance analytics. Leveraging this graph, we develop fundamental approaches to perform a variety of security tasks, such as APT detection, attack scenario reconstruction, and cyber threat-hunting. The proposed technologies use the knowledge of provenance, in terms of causal linkages of activities in an enterprise along with big data techniques to mount a better defense against the insidious attacks that thwart current defenses and afflict enterprises today. In particular, this dissertation includes (1) Sleuth [1], a tag-based approach for attack scenario reconstruction based on prioritizing entities and events by their likeliness to be involved in cyberattacks, (2) Holmes [2], an APT detection system that effectively leverages the correlation between suspicious information flows that arise during an attacker campaign to produce a detection signal that indicates the presence of a coordinated set of suspicious activities, and (3) Poirot [3], a novel inexact graph pattern matching method that assesses an alignment between a query graph extracted from cyber threat intelligence correlations and the provenance graph.

1.1 Dissertation Statement

Our main thesis is that the low-level causal information inferred from kernel audit logs could be utilized to achieve robust and reliable threat detection methods that efficiently pinpoint threats and reveal the high-level picture of attacks by producing compact visual graphs of attack steps.

1.2 Contributions

The contributions of this dissertation are as follows:

- We leverage a compact in-memory dependence graph representation of the kernel audit logs, which serves as the basis for running various high-performance analytics.
- We develop a tag-based approach for identifying subjects, objects and events that are most likely involved in attacks. Tags enable us to prioritize, focus our analysis, and summarize our assessment of the trustworthiness and sensitivity of objects and subjects.
- In order to effectively contain advanced attack campaigns, we develop high-level scenario graphs that provide a very compact, visual summary of the campaign at the moment. Such a summary enables an analyst to quickly ascertain whether there is a significant intrusion, understand how the attacker initially breached security, and determine the impact of the attack.
- We formulates cyber threat-hunting as a graph pattern matching problem to reliably detect known cyberattacks. The matching is conducted based on a novel graph similarity metric which assesses an alignment between a query graph constructed out of Cyber threat intelligence (CTI) correlations and a provenance graph constructed out of kernel audit log records.

1.3 Dissertation Overview

This dissertation includes three main chapters investigating technologies that radically harden enterprise security in three different aspects: SLEUTH introduces a compact in-memory graph representation of kernel audit logs which is deployable in large enterprises and is designed for timely attack scenario reconstruction (Chapter 3), Holmes incorporates signal correlation for APT detection by fusing unreliable loose alerts into a strong detection signal (Chapter 4), and Poirot that searches for the embedding of a certain threat behavior in the provenance graph to uncover the steps of a successful attack campaign (Chapter 5). In the following, we briefly explain each of the three main chapters of this dissertation.

1.3.1 Sleuth: Attack Scenario Reconstruction

Sleuth [1] is a system for reconstruction of attack scenarios on an enterprise host based on real-time analysis of kernel audit logs. To provide an efficient event storage and analysis framework, Sleuth exploits a compact main-memory representation of kernel audit logs. Graph algorithms on main memory representation can be orders of magnitude faster than on-disk representations, an important factor in achieving real-time analysis capabilities. In our experiments, we were able to process 79 hours worth of audit data from a FreeBSD system in 14 seconds, with a main memory usage of 84MB. This performance represents an analysis rate that is 20K times faster than the rate at which the data was generated.

Sleuth uses a tag-based approach to identify entities and events that are most likely involved in attacks. Tags enable us to prioritize and focus our analysis by encoding an assessment of *trustworthiness* and *sensitivity* of data as well as processes. This assessment is based on data provenance derived from audit logs. In this sense, tags derived from audit data are similar to coarse-grain information flow labels. Sleuth leverages a customizable policy framework that can raise detection alerts based on entity tags.

Starting from alerts produced by the attack detection component, our backward analysis algorithm follows the dependencies in the graph to identify the sources of the attack. Starting from the sources, we perform a full impact analysis of the actions of the adversary using a forward search. We present several criteria for pruning these searches in order to produce a compact graph. We also present a number of transformations that further simplify this graph and produce a graph that visually captures the attack in a succinct and semantically meaningful way. Experiments show that our tag-based approach is very effective: for instance, SLEUTH can analyze 38.5M events and produce an attack scenario graph with just 130 events, representing five orders of magnitude reduction in event volume.

1.3.2 HOLMES: Real-time APT Detection

A typical APT consists of a successful penetration (e.g., a drive-by-download or a spear-phishing attack), reconnaissance, command and control (C&C) communication (sometimes using Remote Access Trojans (RATs)), privilege escalation (by exploiting vulnerabilities), lateral movement through the network, exfiltration of confidential information, and so on. This sequence of correlated activities are know as APT kill-chain. The kill-chain provides a reference to understand and map the motivations, targets, and actions of APT actors. Even though the concrete attack steps may vary widely among different APTs, the high-level APT behavior often conforms to the same kill-chain. Our analysis of hundreds of APT reports from [5] suggests that most APTs consist of a subset, if not all, of those steps. More importantly, we make the observation that these steps need to be *causally connected*, and this connectedness is a major indication that an attack is unfolding.

Holmes [2] begins with host audit data and produces a detection signal that maps out the stages of an ongoing APT campaign to the kill chain. To bridge the semantic gap between low-level system-call view and the high-level kill-chain view, we build an intermediate layer based on MITRE's ATT&CK framework [10], which describes close to 200 behavioral patterns defined as *Tactics, Techniques, and Procedures (TTPs)* observed in the wild. Each TTP defines one possible way to realize a particular high-level capability. For instance, the capability of *persistence* in a compromised Linux system can be achieved using 11 distinct TTPs, each of which represents a possible sequence of lower level actions in the ATT&CK framework, e.g., installation of a rootkit, modification of boot scripts, and so on. These lower level actions are closer to the level of abstraction of audit logs, so it is possible to describe TTPs in terms of nodes and edges in the provenance graph.

Taking advantage of inter-TTP information flow dependencies, we correlate various TTPs and map them to high-level APT steps. The final correlation result is summarized in the form of a graph that we call a *High-level Scenario Graph (HSG)*. The HSG provides a compact, visual abstraction of the progress of the campaign at any moment. To distinguish the HSGs that constitute an attack from the benign ones, our approach assigns a severity score to each HSG and raises an alarm if this score bypasses a certain threshold.

1.3.3 Poirot: Cyber Threat-Hunting

Cyber threat intelligence (CTI) is being used to search for indicators of attacks that might have compromised an enterprise network for a long time without being discovered. To have a more effective analysis, CTI open standards have incorporated descriptive relationships showing how the indicators or observables are related to each other. However, these relationships are either completely overlooked in information gathering or not used for threat-hunting. Hence, a vast majority of the current threat-hunting approaches operates only over fragmented views of cyber threats [11, 12], such as signatures (e.g., hashes of artifacts), suspicious file/process names, and IP addresses (domain names).

POIROT [3] uses the relationships between Indicators of Compromise (IOC) artifacts, which contain essential clues on the *behavior* of the attacks inside a compromised system, to uncover the steps of a successful attack campaign. In a nutshell, given a graph-based representation of IOCs and relationships among them that expresses the overall behavior of an APT, which we call a *query graph*, our approach efficiently finds an embedding of this *query graph* in a much larger *provenance graph*, which contains a representation of kernel audit logs over a long period of time. we formulate threat-hunting as a graph pattern matching (GPM) problem searching for causal dependencies or information flows among system entities that are similar to those described in the *query graph*. Our technical approach is based on a novel similarity metric which assesses an alignment between a query graph constructed out of CTI correlations and a provenance graph constructed out of kernel audit log records.

1.4 Dissertation Structure

The organization of the remainder of this dissertation is as follows: In Chapter 2, we provide background material on APT and the defensive approaches and tools. Chapters 3, 4, and 5 present SLEUTH, HOLMES and POIROT, respectively. Chapter 6 highlights the related work and compares it with the work presented in this dissertation. Finally, we provide conclusions on our work and the future research directions in Chapter 7.

CHAPTER 2

BACKGROUND

This chapter includes excerpts and figures from material that is published in [1–4].

In this chapter, we start with providing necessary background information on advanced persistent threats and current defensive approaches. Then, we explain event correlation as a necessary method to confront APT campaigns and mention current approaches. Consequently, we describe kernel audit logs as a source of data containing history of system activities and mention various information flow tracking techniques suitable for event correlation. Finally, we elaborate on Transparent Computing datasets that are used for evaluation of our work. While the material in this chapter is not comprehensive, it gives the reader of this dissertation the required background information to understand the concepts introduced in the following chapters.

2.1 Advanced Persistent Threats

Advanced Persistent Threat is a targeted and stealthy cyberattack that follow a multi-stage threat workflow [13] to break into an enterprise network with the goal of harvesting invaluable information, altering, or destroying critical infrastructures. APT actors are advanced and sophisticated in terms of the resource, tools and knowledge, and they hide their activities between other normal events to persist for a long period of time (weeks, or in some cases, months). The Internet Society's Online Trust Alliance has estimated occurrence of two million cyberattacks

TABLE I

Some of the most devastative cyberattacks, targeting different sectors

Name	**Year**	**Description**
Titan Rain	2003	Targeting US military (called the greatest transfer of wealth in history) stealing blueprints of planes, space-based lasers, missile navigation and nuclear submarines
Stuxnet	2010	Targeting Nuclear Facilities (called first Digital Weapon)
Target	2013	40 million payment card credentials and 70 million customer records lost
Yahoo	2014	Information associated with at least 500 million user accounts was stolen
OPM	2015	Described by federal officials as among the largest breaches of government data in the history of the United States
Deep Panda	2015	Targeting Health Care Services (breach of financial and medical records of up to 80 million customers)
Black Energy	2016	destroyed all available data on the hard drives of a Power Grid Systems
Equifax	2017	Exposed the names, SSN, birth dates, addresses, and, in some instances, driver's license numbers of about 44 percent of the current American population
Marriott	2018	Hundreds of millions of customer records, including credit card and passport numbers, being exfiltrated by the attackers

in 2018, which collectively have resulted in more than $45 billion losses. In addition to financial losses, cyberattacks have the potential to cause life threatening disasters, by targeting critical infrastructures, e.g., PLCs compromised in the Stuxnet worm [14]. Table I represents some of the most devastative cyberattacks, targeting different sectors, since 2003. As shown, business corporations, governmental organizations, religious or educational institutions are all targets of cyberattacks.

In one of the first ever detailed reports on Advanced and Persistent Threats (entitled APT1 [13]), the security firm Mandiant disclosed the goals and activities of a global APT actor. The activities included stealing of hundreds of terabytes of sensitive data (including business plans, technology blueprints, and test results) from at least 141 organizations across a diverse set of industries. They estimated the average duration of persistence of malware in the targeted organizations to be 365 days. Since then, there has been a growing list of documented APTs involving powerful actors, including nation-state actors, on the global scene.

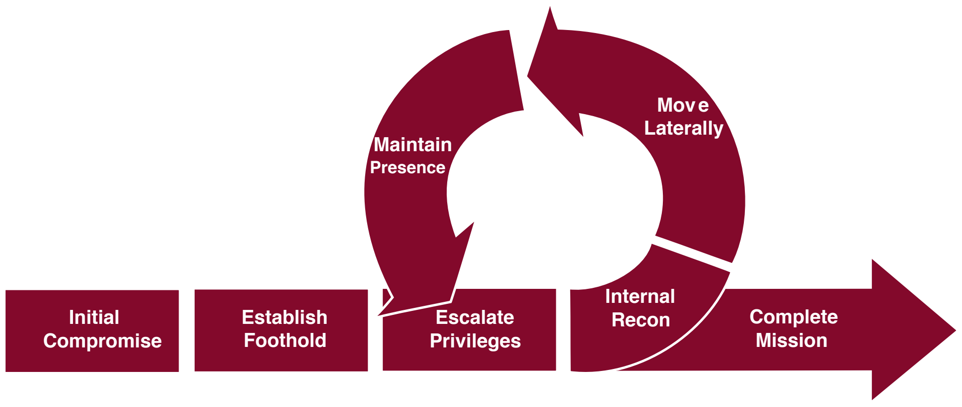


Figure 1. APT Lifecycle.

Understanding the motivations and operations of the APT actors plays a vital role in the challenge of addressing these threats. To further this understanding, the Mandiant report also offered an APT lifecycle model (Figure 1), also known as the *kill-chain*, that allows one to gain perspective on how the APT steps collectively achieve their actors' goals. A typical APT attack consists of a successful penetration (e.g., a drive-by-download or a spear-phishing attack), reconnaissance, command and control (C&C) communication (sometimes using Remote Access Trojans (RATs)), privilege escalation (by exploiting vulnerabilities), lateral movement through the network, exfiltration of confidential information, and so on. In short, the kill-chain provides a reference to understand and map the motivations, targets, and actions of APT actors.

APTs have grown in sophistication since the publication of the first Mandiant report. The details of various exploits used have varied over the years, but the high-level steps have remained mostly the same. While surveying about 300 APT reports [5], we observed that most of the APTs still follow the steps that mostly conform to the kill-chain shown in Figure 1.

2.2 Defensive Approaches

In this section, we introduce different defensive approaches security analysts use to confront cyberattacks. First, we explain intrusion detection techniques, and then, we explain forensics and threat-hunting as two post-attack analysis techniques.

2.2.1 Intrusion Detection

The main goal of Intrusion Detection Systems (IDS) is to find the malicious activities, preferably in real time, and then, block and/or report them to system administrators. There are two broad categories of intrusion detection systems, i.e., host-based and network-based. Network IDS operates on sensors deployed on the network, while host-based IDS operates on basis of log files produced by software running on host computers. Commercially, network-based sensors have proven to be far more viable than host-based sensors. Network sensors can be deployed in just a few locations, e.g., network gateways, yet monitor an entire enterprise network. In contrast, host-based sensors need to be deployed on every host. More importantly, their implementation will differ with the specific OS and software deployed on a host. The drawback of network sensors is that they provide very limited insight into software systems that are both the targets as well as agents of intrusions.

The algorithms and techniques used by IDSes could be divided into three main categories: *misuse detection, anomaly detection,* and *specification-based detection.*

Misuse detection is one of the earliest techniques used by intrusion detection systems to identify infected files. This technique relies on predefined attack signatures that are extracted from known attacks. Consequently, it is relatively easy to interpret the alerts as the matched signatures are associated with specific attack names or identifiers. The biggest limitation of this technique is in discovering unknown threats that do not have a known signature. As cyberattacks become more sophisticated, they may involve zero-day exploits or mutations of known exploits with different signatures, thereby escaping detection.

Anomaly-based intrusion detection is the process of identifying unexpected events which differ from the norm. To this end, a model of normal behavior is learned during a *training phase* in a benign environment. Next, this model is used to detect deviations during the *test phase* in an operation environment. Anomaly detection addresses the main limitation of misuse detection techniques in being able to detect novel attacks. However, their real-world use has been hampered by a relatively high false positive rate, as well as the need for training.

Specification-based intrusion detection focuses on behavior rather than unique signatures of specific attacks. As a result, it is capable of detecting unknown threats. However, this technique has not seen much deployment because it is challenging to define accurate behavioral specifications.

2.2.2 Forensics and Threat-hunting

Forensic analysis is the act of investigating and examining evidences after a detection signal is raised from intrusion detection systems. This process traces evidences related to the incident to provide a more precise description and additional context of an attack. This information would help us in finding the origin of the attack, how the attack is conducted, and the scale of the attack. As a result, security analysts can find the vulnerable services, improve their systems to prevent this attack in future, discover the affected units which are under influence of the attackers, and quarantine or recover these units to stop additional damage from this attack.

When the reports related to an APT is released, a common question that emerges among enterprise security analysts is if their enterprise has been the target of that APT. This process is commonly known as *Threat-hunting*. Answering this question with a high level of confidence often requires lengthy and complicated searches and analysis over host and network logs of the enterprise, recognizing entities that appear in the IOC descriptions among those logs and finally assessing the likelihood that the specific APT successfully infiltrated the enterprise.

Threat-hunting is different from forensic analysis in that it does not start from a detection signal. In threat-hunting, security analysts proactively and iteratively search through the logs to discover the attacks which have evaded the existing security solutions. These searches usually look for IOCs that with high confidence can indicate the occurrence of an attack, such as hash values of malware files, low reputation IP addresses and domain names, or unique names used by certain attackers.

2.3 Event Correlation

As APT campaigns employ multiple attack vectors, it poses a great challenge on defensive approaches which have isolated views. To derive a high-level picture of an ongoing APT campaign, events coming from heterogeneous sources are required to be processed and correlated. Enterprises today, are capable of gathering an extremely rich set of telemetry data from different sources, such as firewalls, IPS/IDS, endpoint and server security systems, identity and access management tools, application security tools, application firewalls and database firewalls. However, there is limited capabilities in relating or mapping these sources together. In particular, it is difficult for a security analyst to distinguish truly significant attacks — the proverbial “needle-in-a-haystack” — from background noise, and it remains a major challenge to accurately “connect the dots”, i.e., piece together fragments of an attack campaign that span multiple applications or hosts and extend over a long time period.

Security Information and Event Management (SIEM) systems aggregate security-related events and alerts from multiple sources and make reasoning by correlating them. Some examples of these systems are Splunk [15], LogRhythm [16] and IBM QRadar [17]. Research in this area falls into the following main categories [18]:

- *Statistical analysis:* In this approach [19], successive steps of a multi-step attack are correlated based on statistical features of alerts that occur temporally close to each other. However, as attacks are very rare events and the amount of available training data is often limited, the managing of false positives is very difficult.

- *Precondition-based analysis:* In this method, two attacks are correlated if the post-condition of an attack contributes to the precondition of a subsequent attack [20]. However, this type of reasoning requires attack models (describing pre- and post-conditions) to be developed which might not be always practical.

- *Graph-based analysis:* This technique models the entities involved in a coordinated set of attack stages in a graph structure. Then, various graph analytics could be applied for alarm correlation [21], such as spectral clustering and page rank. It is also possible to correlate alarms based on learning how predictable a correlation between two events might be [22].

In this dissertation, we use kernel audit logs as an invaluable source to piece together the causal chain of events and activities. In the next section, we provide necessary background information about the kernel audit logs.

2.4 Kernel Audit Logs

Applications do not have permission to directly access system resources such as CPU, memory, or Input/Output (I/O). The kernel of the operating system acts as a bridge between software applications and system resources, and applications should requests a service from the kernel. These requests, which are called system calls, are processed by the kernel and a response is sent to the applications. The kernel has the ability to intercept every system call and record them, and there are different tools that could be used for this purpose. For example, `auditd` can be the source of audit data in Linux, while it can be `dtrace` for BSD, and ETW for Windows. To be consistent and independent of the low-level system details, we collect and process this raw audit data into an OS-neutral format. This is the input format that we use for the systems proposed in this dissertation. This input captures events relating to principals (users), files (e.g., operations for I/O, file creation, ownership, and permission), memory (e.g., mprotect and mmap) processes (e.g., creation, and privilege change), and network connections.

To provide a convenient formalism for reasoning about system activity and allow tracking causality and information flow efficiently, the kernel audit logs could be represented as a graph,

TABLE II

System event types

Purpose	**Relevant System Calls**
Information Flow	*clone* (process), *fork* , *msgsnd*, *msgrcv*, *write*, *send*, *read*, *recv*, *exec*
Creation	*open*, *creat*, *dup*, *link*, *socket*, *socketpair*
Preparatory	*lseek*, *connect*, *listen*, *accept*, *bind*, *link*
Termination	*close*, *exit*, *exit_group*, *unlink*, *kill*

called the provenance graph, which is a labeled, typed, and directed graph. In this graph, nodes represent system entities, which have different types such as files and processes, while edges represent information flow and causal dependency among these nodes taking into account the direction. Not all the system calls indicate an information flow or a causal dependency, and therefore are not required to be collected. For example, Table II shows some of the most important security related system calls that are needed for a BSD operating system. In this table, we show different categories of system calls according to their purpose. Some system calls are responsible for the actual information flow between objects. For instance, when a new process is created via a `clone` system call, it inherits the file descriptors of its parent. Therefore, there is information flow from the parent to the child process. A subset of the system calls (third row of Table II) is responsible for initializing and setting up data structures rather than dealing with information flow directly. For example, the *socketpair* system call creates two sockets. *Preparatory* system calls initialize data structures, and in certain cases provide the provenance of the subsequent data. For example, by checking the *lseek* system call and considering file offsets, we only track specific offsets of a file to prevent unnecessary dependencies. *Termination* system calls deal with the destruction of objects.

TABLE III

Information flow events.

Relevant System Calls	**From**	**To**	**Source**	**Destination**
clone (process), *fork, vfork, rfork, msgsnd*	Process	Process	event caller	arg(s)
wait, msgrcv	Process	Process	arg(s)	event caller
write, pwrite, writev, pwritev, send, sendto, sendmsg	Process	File/Socket	event caller	arg(s)
read, recv, recvfrom, recvmsg, execl, execv, execle, execve, execlp, execvp	File/Socket	Process	arg(s)	event caller

To construct a provenance graph from the sequence of system calls, we need to extract the process initiating the system call, the type of the system call, and their arguments. Table III shows some examples of how certain system calls are converted to nodes and edges of a provenance graph. The second and third columns show the direction of the edge and the types of nodes (File, socket, process), while the last two columns show how the node names and attributes are extracted from the system call event. we use *arg(s)* to indicate the argument(s) of system calls to refer to the object(s) that the *caller process* manipulates. In particular, depending on the system call, the argument type may be the *id* of a process, the *name* of a file, or a *descriptor* referring to a file/socket. As shown in the table, there are different kinds of information flow between system objects. These include: (*i*) from a process to another process initiated by events like *fork* and *clone*, which indicates a causal dependency between them, (*ii*) from a process to a file/socket initiated by events like *write* and *send*, and (*iii*) from a file/socket to a process initiated by events such as *read*, and *receive*, which indicate an information flow.

Figure 2 shows a snippet of a provenance graph. The graph on the left shows the "SSH" daemon process which has created two "bash" processes, and the graph on the right is generated as the result of Firefox browser downloading an executable file ("Openme.exe") from an IP

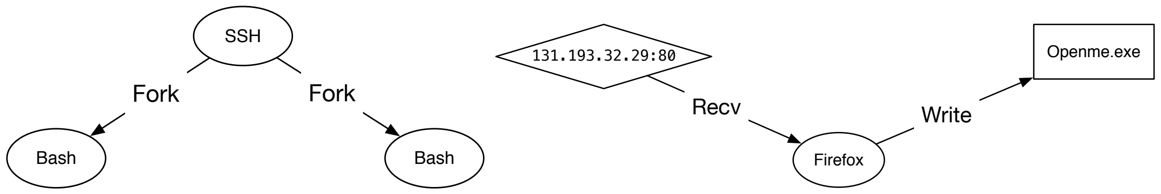


Figure 2. Sample provenance graph snippets

address ("131.193.32.29:80"). The direction of edges show that the two "bash" processes are children of "SSH" and are causally dependent to it. Also, the direction indicates the flow of information originating from the socket connection to the downloaded file. To be consistent, we use certain node shapes to distinguish entity types. For this figure and the rest of this dissertation, Ovals, diamonds, rectangles, and stars represent processes, sockets, files, and users, respectively. In addition, memory objects and registry entries are both represented by pentagons. As it is shown, the provenance graphs leverage the full historical context of a system and can include hundreds of millions/billions nodes.

2.5 Information Flow Analysis

The high-level idea in information flow analysis is to tag (taint) an interesting data and track it as it propagates through the system. Information flow analysis could be done in different granularities, i.e., fine-grained and coarse-grained. In fine-grained information flow analysis [23–25], a shadow memory is used that keeps a tag for every word in memory. When the tainted data influence the value of some memory region, those parts get tainted with the tag bits. This type of tracking typically requires instrumenting applications (e.g., using Pin [26]) and is infeasible because of too high overhead (usually around 10×-20×).

On the contrary, coarse-grained information flow analysis mainly focuses on system call data (kernel audit logs) which its collection imposes low runtime overhead (around 2%). In particular, one can initiate forward or backward graph traversals on a provenance graph built from the kernel audit logs to investigate cyberattacks. For example, after an attack is detected, system analysts use the detection point as a seed to initiate backward tracking strategies to determine the root-cause of that attack. Similarly, system analysts can initiate forward tracking methods to find out the impacts of the attack and the entities that are affected.

The lower overhead of coarse-grained information flow analysis comparing to the fine-grained one comes at the cost of losing the precision, and sometimes, the coarse granularity of system calls may limit reasoning about information flows. For example, when a process reads multiple encrypted files and writes the decrypted content to multiple files in random order, knowing which of the decrypted files corresponds to which one of the encrypted ones is a challenge that cannot be overcome without fine-grained and memory-level information flow tracking.

2.6 Transparent Computing Datasets

The work presented in this dissertation is evaluated against datasets released by Transparent Computing (TC) program for adversarial engagements. The TC program [27] is a DARPA effort aiming to add more transparency to computing systems by using minimal overhead components to monitor interactions during system operation. This program involved multiple red-team, blue-team adversarial engagements which some are publicly available [28].

In these engagements, the red-team has been responsible for simulating multiple cyberattacks on a network consisting of different platforms, such as Linux, FreeBSD, and Windows.

Before the engagements, each machine was set up with some vulnerable software that later gets exploited. During the engagements, kernel audit logs were being collected, processed, and converted to a common data representation for ease of analysis[1]. To further mix normal and attack logs, the red-team also performed benign activities on the target hosts in parallel with attacks. Routine system activities include, but are not limited to, web browsing, checking email, software upgrading, administrative tasks using PowerShell (in Windows), running programs that require administrative privileges, and so on. Collectively, these engagements involve months of audit log collection activity, while attacks constitute less than 0.001% of the audit data volume.

[1]While some additional finer-granularity data might be provided in some platforms (such as Linux dataset containing unit abstraction data [25]), we have not considered them in our evaluation.

CHAPTER 3

SLEUTH: REAL-TIME ATTACK SCENARIO RECONSTRUCTION FROM COTS AUDIT DATA

This chapter includes excerpts and figures from material that is published in [1].

This chapter presents SLEUTH, a system that consumes kernel audit logs from different operating systems and process them into a platform-neutral graph representation in memory. Both graph representation and in-memory storage provide a basis suitable for running high performance computations on kernel audit logs. This graph representation serves as the basis for the systems presented in the next chapters (HOLMES and POIROT) of this dissertation. In addition to proposing an efficient in-memory graph representation, SLEUTH can also alert analysts in real-time about an ongoing campaign, and provide them with a compact, visual summary of the activity in seconds or minutes after the attack. This would enable a timely response before enormous damage is inflicted on the victim enterprise.

3.1 Introduction

Using kernel audit logs for real-time analytics such as attack detection poses the following additional challenges over a purely forensic analysis:

1. *Event storage and analysis:* How can we store the millions of records from event streams efficiently and have algorithms sift through this data in a matter of seconds?

2. *Prioritizing entities for analysis:* How can we assist the analyst, who is overwhelmed with the volume of data, prioritize and quickly "zoom in" on the most likely attack scenario?

3. *Scenario reconstruction:* How do we succinctly summarize the attack scenario, starting from the attacker's entry point and identifying the impact of the entire campaign on the system?

4. *Dealing with common usage scenarios:* How does one cope with normal, benign activities that may resemble activities commonly observed during attacks, e.g., software downloads?

5. *Fast, interactive reasoning:* How can we provide the analyst with the ability to efficiently reason through the data, say, with an alternate hypothesis?

Figure 3 provides an overview of our approach. SLEUTH assumes that attacks initially come from outside the enterprise. For example, an adversary could start the attack by hijacking a web browser through externally supplied malicious input, by plugging in an infected USB memory stick, or by supplying a zero-day exploit to a network server running within the enterprise. We assume that the adversary *has not* implanted persistent malware on the host *before* SLEUTH started monitoring the system. We also assume that the OS kernel and audit systems are trustworthy.

The first contribution of SLEUTH, which addresses the challenge of efficient event storage and analysis, is the development of a compact main-memory dependence graph representation (Section 3.2). Graph algorithms on main memory representation can be orders of magnitude faster than on-disk representations, an important factor in achieving real-time analysis capabilities. In our experiments, we were able to process 79 hours worth of audit data from a FreeBSD

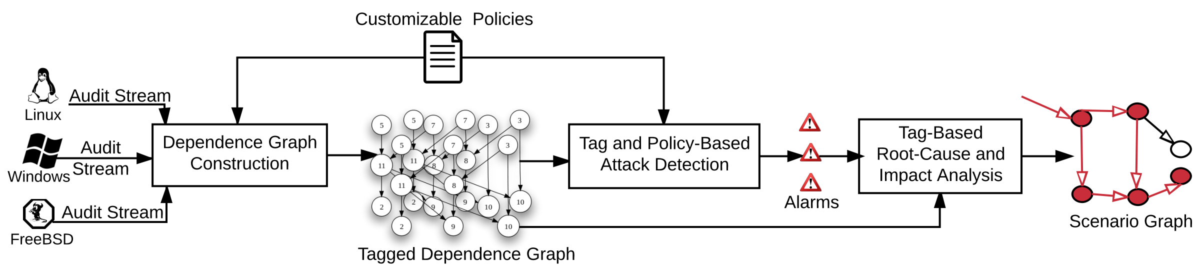


Figure 3. SLEUTH System Overview

system in 14 seconds, with a main memory usage of 84MB. This performance represents an analysis rate that is 20K times faster than the rate at which the data was generated.

The second major contribution of SLEUTH is the development of a tag-based approach for identifying subjects, objects and events that are most likely involved in attacks. Tags enable us to prioritize and focus our analysis, thereby addressing the second challenge mentioned above. Tags encode an assessment of *trustworthiness* and *sensitivity* of data (i.e., objects) as well as processes (subjects). This assessment is based on data provenance derived from audit logs. In this sense, tags derived from audit data are similar to coarse-grain information flow labels. Our analysis can naturally support finer-granularity tags as well, e.g., fine-grained taint tags [29, 30], if they are available. Tags are described in more detail in Section 3.3, together with their application to attack detection.

A third contribution of SLEUTH is the development of novel algorithms that leverage tags for root-cause identification and impact analysis (Section 3.5). Starting from alerts produced by the attack detection component shown in Figure 3, our backward analysis algorithm follows the dependencies in the graph to identify the sources of the attack. Starting from the sources, we perform a full impact analysis of the actions of the adversary using a forward search. We present several criteria for pruning these searches in order to produce a compact graph. We also present a number of transformations that further simplify this graph and produce a graph that visually captures the attack in a succinct and semantically meaningful way, e.g., the graph in Figure 5. Experiments show that our tag-based approach is very effective: for instance, SLEUTH can analyze 38.5M events and produce an attack scenario graph with just 130 events, representing five orders of magnitude reduction in event volume.

The fourth contribution of SLEUTH, aimed at tackling the last two challenges mentioned above, is a customizable policy framework (Section 3.4) for tag initialization and propagation. Our framework comes with sensible defaults, but they can be overridden to accommodate behaviors specific to an OS or application. This enables tuning of our detection and analysis techniques to avoid false positives in cases where benign applications exhibit behaviors that resemble attacks. (See Section 3.6.6 for details.) Policies also enable an analyst to test out “alternate hypotheses” of attacks, by reclassifying what is considered trustworthy or sensitive and re-running the analysis. If an analyst suspects that some behavior is the result of an attack, they can also use policies to capture these behaviors, and rerun the analysis to discover its cause and impact. Since we can process and analyze audit data tens of thousands of times faster than the rate at which it is generated, efficient, parallel, real-time testing of alternate hypotheses is possible.

The final contribution of Sleuth is an experimental evaluation (Section 3.6), based mainly on a TC dataset (described in Section 2.6). In this evaluation, attack campaigns resembling modern APTs were carried out on Windows, FreeBSD and Linux hosts over a two week period. In this evaluation, Sleuth was able to:

- process, in a matter of seconds, audit logs containing tens of millions of events generated during the engagement;
- successfully detect and reconstruct the details of these attacks, including their entry points, activities in the system, and exfiltration points;
- filter away extraneous events, achieving very high reductions rates in the data (up to 100K times), thus providing a clear semantic representation of these attacks containing almost no noise from other activities in the system; and
- achieve low false positive and false negative rates.

Our evaluation is not intended to show that we detected the most sophisticated adversary; instead, our point is that, given several unknown possibilities, the prioritized results from our system can be right on spot in real-time, without any human assistance. Thus, it really fills a gap that exists today, where forensic analysis seems to be primarily initiated manually.

3.2 Main Memory Dependency Graph

To support fast detection and real-time analysis, we store dependencies in a graph data structure. One possible option for storing this graph is a graph database. However, the performance [31] of popular databases such as Neo4J [32] or Titan [33] is limited for many graph algorithms unless main memory is large enough to hold most of data. Moreover, the memory use of general graph databases is too high for our problem. Even STINGER [34] and NetworkX [35], two graph databases optimized for main-memory performance, use about 250 bytes and 3KB, respectively, per graph edge [31]. The number of audit events reported on enterprise networks can easily range in billions to tens of billions per day, which will require main memory in the range of several terabytes. In contrast, we present a much more space-efficient dependence graph design that uses only about 10 bytes per edge. In one experiment, we were able to store 38M events in just 329MB of main memory.

The dependency graph is a per-host data structure. It can reference entities on other hosts but is optimized for the common case of intra-host reference. The graph represents two types of entities: *subjects*, which represent processes, and *objects*, which represent entities such as files, pipes, and network connections. Subject attributes include process id (pid), command line, owner, and tags for code and data. Objects attributes include name, type (file, pipe, socket, etc.), owner, and tags.

Events reported in the audit log are captured using labeled edges between subjects and objects or between two subjects. For brevity, we use UNIX names such as `read`, `connect`, and `execve` for events. A detailed explanation of the techniques that are developed to reduce storage requirements of SLEUTH was later published in [36].

3.3 Tags and Attack Detection

We use tags to summarize our assessment of the trustworthiness and sensitivity of objects and subjects. This assessment can be based on three main factors:

- *Provenance:* the tags on the immediate predecessors of an object or subject in the dependence graph,
- *Prior system knowledge:* our knowledge about the behavior of important applications, such as remote access servers and software installers, and important files such as `/etc/passwd` and `/dev/audio`, and
- *Behavior:* observed behavior of subjects, and how they compare to their expected behavior.

We have developed a policy framework, described in Section 3.4, for initializing and propagating tags based on these factors. In the absence of specific policies, a default policy is used that propagates tags from inputs to outputs. The default policy assigns to an output the lowest among the trustworthiness tags of the inputs, and the highest among the confidentiality tags. This policy is conservative: it can err on the side of over-tainting, but will not cause attacks to go undetected, or cause a forward (or backward) analysis to miss objects, subjects or events.

Tags play a central role in SLEUTH. They provide important context for attack detection. Each audited event is interpreted in the context of these tags to determine its likelihood of contributing to an attack. In addition, tags are instrumental for the speed of our forward and backward analysis. Finally, tags play a central role in scenario reconstruction by eliminating vast amounts of audit data that satisfy the technical definition of dependence but do not meaningfully contribute to our understanding of an attack.

3.3.1 Tag Design

We define the following *trustworthiness tags* (*t-tags*):

- *Benign authentic* tag is assigned to data/code received from sources trusted to be benign, and whose authenticity can be verified.
- *Benign* tag reflects a reduced level of trust than benign authentic: while the data/code is still believed to be benign, adequate authentication hasn't been performed to verify the source.
- *Unknown* tag is given to data/code from sources about which we have no information on trustworthiness. Such data *can sometimes be* malicious.

Policies define what sources are benign and what forms of authentication are sufficient. In the simplest case, these policies take the form of whitelists, but we support more complex policies as well. If no policy is applicable to a source, then its t-tag is set to *unknown.*

We define the following *confidentiality tags* (*c-tags*), to reason about information stealing attacks:

- *Secret:* Highly sensitive information, such as login credentials and private keys.
- *Sensitive:* Data whose disclosure can have a significant security impact, e.g., reveal vulnerabilities in the system, but does not provide a direct way for an attacker to gain access to the system.
- *Private:* Data whose disclosure is a privacy concern, but does not necessarily pose a security threat.
- *Public:* Data that can be widely available, e.g., on public web sites.

An important aspect of our design is the separation between t-tags for code and data. Specifically, a subject (i.e., a process) is given two t-tags: one that captures its *code trustworthiness* (code t-tag) and another for its *data trustworthiness* (data t-tag). This separation significantly improves attack detection. More importantly, it can significantly speed up forensic analysis by focusing it on fewer suspicious events, while substantially reducing the size of the reconstructed scenario. Note that confidentiality tags are associated only with data (and not code).

Pre-existing objects and subjects are assigned initial tags using *tag initialization policies.* Objects representing external entities, such as a remote network connection, also need to be assigned initial tags. The rest of the objects and subjects are created during system execution, and their tags are determined using *tag propagation policies.* Finally, attacks are detected using behavior-based policies called *detection policies.*

As mentioned before, if no specific policy is provided, then sources are tagged with *unknown* trustworthiness. Similarly, in the absence of specific propagation policies, the default conservative propagation policy is used.

3.3.2 Tag-based Attack Detection

An important constraint in SLEUTH is that we are limited to information available in audit data. This suggests the use of provenance reflected in audit data as a possible basis for detection. Since tags are a function of provenance, we use them for attack detection. Note that in our threat model, audit data is trustworthy, so tags provide a sound basis for detection.

A second constraint in SLEUTH is that detection methods should not require detailed application-specific knowledge. In contrast, most existing intrusion detection and sandboxing techniques interpret each security-sensitive operation in the context of a specific application to determine whether it could be malicious. This requires expert knowledge about the application, or in-the-field training in a dynamic environment, where applications may be frequently updated.

Instead of focusing on application behaviors that tend to be variable, we focus our detection techniques on the high-level objectives of most attackers, such as backdoor insertion and data exfiltration. Specifically, we combine reasoning about an attacker's *motive* and *means.* If an event in the audit data can help the attacker achieve his/her key high-level objectives, that would provide the motivation and justification for using that event in an attack. But this is not enough: the attacker also needs the means to cause this event, or more broadly, influence it. Note that our tags are designed to capture means: if a piece of data or code bears the *unknown* t-tag, then it was derived from (and hence influenced by) untrusted sources.

As for the high-level objectives of an attacker, several reports and white papers have identified that the following steps are typical in most advanced attack campaigns [5, 13, 37]:

1. Deploy and run attacker's code on victim system.
2. Replace or modify important files, e.g., `/etc/passwd` or ssh keys.
3. Exfiltrate sensitive data.

Attacks with a transient effect may be able to avoid the first two steps, but most sophisticated attacks, such as those used in APT campaigns, require the establishment of a more permanent footprint on the victim system. In those cases, there does not seem to be a way to avoid one or both of the first two steps. Even in those cases where the attacker's goal could be achieved without establishing a permanent base, the third step usually represents an essential attacker goal.

Based on the above reasoning, we define the following policies for attack detection that incorporate the attacker's objectives and means:

- *Untrusted code execution:* This policy triggers an alarm when a subject with a higher code t-tag executes (or loads) an object with a lower t-tag[1].
- *Modification by subjects with lower code t-tag:* This policy raises an alarm when a subject with a lower code t-tag modifies an object with a higher t-tag. Modification may pertain to the file content or other attributes such as name, permissions, etc.
- *Confidential data leak:* An alarm is raised when untrusted subjects exfiltrate sensitive data. Specifically, this policy is triggered on network writes by subjects with a *sensitive* c-tag and a code t-tag of *unknown.*

[1]Customized policies can be defined for interpreters such as `bash` so that reads are treated the same as loads.

- *Preparation of untrusted data for execution:* This policy is triggered by an operation by a subject with a code t-tag of *unknown,* provided this operation makes an object executable. Such operations include `chmod` and `mprotect`[1,2].

It is important to note that "means" is not diluted just because data or code passes through multiple intermediaries. For instance, the untrusted code policy does not require a direct load of data from an unknown web site; instead, the data could be downloaded, extracted, uncompressed, and possibly compiled, and then loaded. Regardless of the number of intermediate steps, this policy will be triggered when the resulting file is loaded or executed. This is one of the most important reasons for the effectiveness of our attack detection.

Today's vulnerability exploits typically do not involve untrusted code in their first step, and hence won't be detected by the untrusted code execution policy. However, the eventual goal of an attacker is to execute his/her code, either by downloading and executing a file, or by adding execute permissions to a memory page containing untrusted data. In either case, one of the above policies can detect the attack. A subsequent backward analysis can help identify the first step of the exploit.

Additional detector inputs can be easily integrated into SLEUTH. For instance, if an external detector flags a subject as a suspect, this can be incorporated by setting the code t-tag of the

[1]Binary code injection attacks on today's OSes ultimately involve a call to change the permission of a writable memory page so that it becomes executable. To the extent that such memory permission change operations are included in the audit data, this policy can spot them.

[2]Our implementation can identify `mprotect` operations that occur in conjunction with library loading operations. This policy is not triggered on those `mprotect`'s.

subject to *unknown.* As a result, the remaining detection policies mentioned above can all benefit from the information provided by the external detector. Moreover, setting of *unknown* t-tag at suspect nodes preserves the dependency structure between the graph vertices that cause alarms, a fact that we exploit in our forensic analysis.

The fact that many of our policies are triggered by untrusted code execution should not be interpreted to mean that they work in a static environment, where no new code is permitted in the system. Indeed, we expect software updates and upgrades to be happening constantly, but in an enterprise setting, we don't expect end users to be downloading unknown code from random sites. Accordingly, we subsequently describe how to support standardized software updating mechanisms such as those used on contemporary OSes.

3.4 Policy Framework

We have developed a flexible policy framework for tag assignment, propagation, and attack detection. We express policies using a simple rule-based notation, e.g.,

$$exec(s, o):\ o.ttag < benign \rightarrow alert(\texttt{"UntrustedExec"})$$

This rule is triggered when the subject s executes a (file) object o with a t-tag less than *benign*. Its effect is to raise an alert named `UntrustedExec`. As illustrated by this example, rules are generally associated with events, and include conditions on the attributes of objects and/or subjects involved in the event. Attributes of interest include:

- *name:* regular expressions can be used to match object names and subject command lines. We use Perl syntax for regular expressions.

- *tags:* conditions can be placed on t-tags and c-tags of objects and/or subjects. For subjects, code and data t-tags can be independently accessed.
- *ownership and permission:* conditions can be placed on the ownership of objects and subjects, or permissions associated with the object or the event.

The effect of a policy depends on its type. The effect of a detection policy is to raise an alarm. For tag initialization and propagation policies, the effect is to modify tag(s) associated with the object or subject involved in the event. While we use a rule-based notation to specify policies in this chapter, in our implementation, each rule is encoded as a (C++) function.

To provide a finer degree of control over the order in which different types of policies are checked, we associate policies with *trigger points* instead of events. In addition, trigger points provide a level of indirection that enables sharing of policies across distinct events that have a similar purpose. Table IV shows the trigger points currently defined in our policy framework. The first column identifies events, the second column specifies the direction of information flow, and the last two columns define the trigger points associated with these events.

Note that we use a special event called `define` to denote audit records that define a new object. This pseudo-event is assumed to have occurred when a new object is encountered for the first time, e.g., establishment of a new network connection, the first mention of a pre-existing file, creation of a new file, etc. The remaining events in the table are self-explanatory.

When an event occurs, all detection policies associated with its alarm trigger are executed. Unless specifically configured, detection policies are checked only when the tag of the target subject or object is about to change. ("Target" here refers to the destination of data flow in an operation.) Following this, policies associated with the event's tag triggers are tried in the order in which they are specified. As soon as a matching rule is found, the tags specified by this rule are assigned to the target of the event, and the remaining tag policies are not evaluated.

Our current detection policies are informally described in the previous section. We therefore focus in this section on our current tag initialization and propagation policies.

3.4.1 Tag Initialization Policies

These policies are invoked at the *init* trigger, and are used to initialize tags for new objects, or preexisting objects when they are first mentioned in the audit data. Recall that when a subject creates a new object, the object inherits the subject's tags by default; however, this can be overridden using tag initialization policies.

Our current tag initialization policy is as follows. Note the use of regular expressions to conveniently define initial tags for groups of objects.

$$\mathit{init}(o)\colon\ \mathit{match}(o.\mathit{name}, \texttt{"\^{}IP:(10\textbackslash.0|127)"}) \rightarrow o.\mathit{ttag} = \texttt{BENIGN\_AUTH}, o.\mathit{ctag} = \texttt{PRIVATE}$$

$$\mathit{init}(o)\colon\ \mathit{match}(o.\mathit{name}, \texttt{"\^{}IP:"}) \rightarrow o.\mathit{ttag} = \texttt{UNKNOWN}, o.\mathit{ctag} = \texttt{PRIVATE}$$

TABLE IV

Edges with policy trigger points. In the direction column, S indicates subject, and O indicates object. The next two columns indicate trigger points for detection policies and tag setting policies.

Event	Direction	Alarm trigger	Tag trigger
`define`			*init*
`read`	O→S	*read*	*propRd*
`load, execve`	O→S	*exec*	*propEx*
`write`	S→O	*write*	*propWr*
`rm, rename`	S→O	*write*	
`chmod, chown`	S→O	*write, modify*	
`setuid`	S→S		*propSu*

$$\mathit{init}(o)\colon\ o.\mathit{type} == \texttt{FILE} \rightarrow o.\mathit{ttag} = \texttt{BENIGN\_AUTH}, o.\mathit{ctag} = \texttt{PUBLIC}$$

The first rule specifies tags for intranet connections, identified by address prefixes 10.0 and 127 for the remote host. It is useful in a context where SLEUTH isn't deployed on the remote host[1]. The second rule states that all other hosts are untrusted. All preexisting files are assigned the same tags by the third rule. Our implementation uses two additional policies that specify c-tags.

3.4.2 Tag Propagation Policies

These policies can be used to override default tag propagation semantics. Different tag propagation policies can be defined for different groups of related event types, as indicated in the "Tag trigger" column in Table IV.

Tag propagation policies can be used to prevent "over-tainting" that can result from files such as `.bash_history` that are repeatedly read and written by an application each time it is invoked. The following policy skips taint propagation for this specific file:

$$\mathit{propRd}(s, o)\colon\ \mathit{match}(o.\mathit{name}, \texttt{"\textbackslash.bash\_history\$"}) \rightarrow \mathit{skip}^{2}$$

Here is a policy that treats files read by `bash`, which is an interpreter, as a load, and hence updates the code t-tag.

[1]If SLEUTH is deployed on the remote host, there will be no `define` event associated with the establishment of a network connection, and hence this policy won't be triggered. Instead, we will already have computed a tag for the remote network endpoint, which will now propagate to any local subject that reads from the connection.

[2]Here, "skip" means do nothing, i.e., leave tags unchanged.

$$\mathit{propRd}(s,o)\colon\ \mathit{match}(s.\mathit{cmdline}, \texttt{"\^{}/bin/bash\$"}) \rightarrow s.\mathit{code\_ttag} = s.\mathit{data\_ttag} = o.\mathit{ttag}, s.\mathit{ctag} = o.\mathit{ctag}$$

Although trusted servers such as `sshd` interact with untrusted sites, they can be expected to protect themselves, and let only authorized users access the system. Such servers should not have their data trustworthiness downgraded. A similar comment applies to programs such as software updaters and installers that download code from untrusted sites, but verify the signature of a trusted software provider before the install.

$$\mathit{propRd}(o,s)\colon\ \mathit{match}(s.\mathit{cmdline}, \texttt{"\^{}/sbin/sshd\$"}) \rightarrow \mathit{skip}$$

Moreover, when the login phase is complete, typically identified by execution of a `setuid` operation, the process should be assigned appropriate tags.

$$\mathit{propSu}(s)\colon$$

$$\mathit{match}(s.\mathit{cmdline}, \texttt{"\^{}/usr/sbin/sshd\$"}) \rightarrow s.\mathit{code\_ttag} = s.\mathit{data\_ttag} = \texttt{BENIGN}, s.\mathit{ctag} = \texttt{PRIVATE}$$

3.5 Tag-Based Bi-Directional Analysis

3.5.1 Backward Analysis

The goal of backward analysis is to identify the entry points of an attack campaign. Entry points are the nodes in the graph with an in-degree of zero and are marked untrusted. Typically they represent network connections, but they can also be of other types, e.g., a file on a USB stick that was plugged into the victim host.

The starting points for the backward analysis are the alarms generated by the detection policies. In particular, each alarm is related to one or more entities, which are marked as suspect nodes in the graph. Backward search involves a backward traversal of the graph to identify paths that connect the suspect nodes to entry nodes. We note that the direction of the dependency edges is reversed in such a traversal and in the following discussions. Backward search poses several significant challenges:

- *Performance:* The dependence graph can easily contain hundreds of millions of edges. Alarms can easily number in thousands. Running backward searches on such a large graph is computationally expensive.
- *Multiple paths:* Typically numerous entry points are backward reachable from a suspect node. However, in APT-style attacks, there is often just one real entry point. Thus, a naive backward search can lead to a large number of false positives.

The key insight behind our approach is that tags can be used to address both challenges. In fact, tag computation and propagation is already an implicit path computation, which can be reused. Furthermore, a tag value of *unknown* on a node provides an important clue about the likelihood of that node being a potential part of an attack. In particular, if an *unknown* tag exists for some node A, that means that there exists at least a path from an untrusted entry node to node A, therefore node A is more likely to be part of an attack than other neighbors with *benign* tags. Utilizing tags for the backward search greatly reduces the search space by eliminating many irrelevant nodes and sets SLEUTH apart from other scenario reconstruction approaches such as [8, 25].

Based on this insight, we formulate backward analyis as an instance of shortest path problem, where tags are used to define edge costs. In effect, tags are able to “guide” the search along relevant paths, and away from unlikely paths. This factor enables the search to be completed without necessarily traversing the entire graph, thus addressing the performance challenge. In addition, our shortest path formulation addresses the multiple paths chalenge by by preferring the entry point closest (as measured by path cost) to a suspect node.

For shortest path, we use Dijkstra's algorithm, as it discovers paths in increasing order of cost. In particular, each step of this algorithm adds a node to the shortest path tree, which consists of the shortest paths computed so far. This enables the search to stop as soon as an entry point node is added to this tree.

Cost function design. Our design assigns low costs to edges representing dependencies on nodes with *unknown* tags, and higher costs to other edges. Specifically, the costs are as follows:

- Edges that introduce a dependency from a node with *unknown* code or data t-tag to a node with *benign* code or data t-tag are assigned a cost of 0.
- Edges introducing a dependency from a node with *benign* code and data t-tags are assigned a high cost.
- Edges introducing dependencies between nodes already having an *unknown* tag are assigned a cost of 1.

The intuition behind this design is as follows. A benign subject or object immediately related to an `unknown` subject/object represents the boundary between the malicious and benign portions of the graph. Therefore, they must be included in the search, thus the cost of these edges is 0.

Information flows among benign entities are not part of the attack, therefore we set their cost to very high so that they are excluded from the search. Information flows among untrusted nodes are likely part of an attack, so we set their cost to a low value. They will be included in the search result unless alternative paths consisting of fewer edges are available.

3.5.2 Forward Analysis

The purpose of forward analysis is to assess the impact of a campaign, by starting from an entry point and discovering all the possible effects dependent on the entry point. Similar to backward analysis, the main challenge is the size of the graph. A naive approach would identify and flag all subjects and objects reachable from the entry point(s) identified by backward analysis. Unfortunately, such an approach will result in an impact graph that is too large to be useful to an analyst. For instance, in our experiments, a naive analysis produced impact graphs with millions of edges, whereas our refined algorithm reduces this number by 100x to 500x.

A natural approach for reducing the size is to use a distance threshold d_{th} to exclude nodes that are "too far" from the suspect nodes. Threshold d_{th} can be interactively tuned by an analyst. We use the same cost metric that was used for backward analysis, but modified to consider confidentiality[1]. In particular, edges between nodes with high confidentiality tags (e.g., *secret*) and nodes with low code integrity tags (e.g., *unknown* process) or low data integrity tags (e.g., *unknown* socket) are assigned a cost of 0, while edges to nodes with *benign* tags are assigned a high cost.

[1]Recall that some alarms are related to exfiltration of confidential data, so we need to decide which edges representing the flow of confidential information should be included in the scenario.

3.5.3 Reconstruction and Presentation

We apply the following simplifications to the output of forward analysis, in order to provide a more succinct view of the attack:

- *Pruning uninteresting nodes.* The result of forward analysis may include many dependencies that are not relevant for the attack, e.g., subjects writing to cache and log files, or writing to a temporary file and then removing it. These nodes may appear in the results of the forward analysis but no suspect nodes depend on them, so they can be pruned.
- *Merging entities with the same name.* This simplification merges subjects that have the same name, disregarding their process ids and command-line arguments.
- *Repeated event filtering.* This simplification merges into one those events that happen multiple times (e.g., multiple writes, multiple reads) between the same entities. If there are interleaving events, then we show two events representing the first and the last occurrence of an event between the two entities.

3.6 Experimental Evaluation

3.6.1 Implementation

Most components of Sleuth, including the graph model, policy engine, attack detection and some parts of the forensic analysis are implemented in C++, and consist of about 9.5KLoC. The remaining components, including that for reconstruction and presentation, are implemented in Python, and consist of 1.6KLoC.

3.6.2 Data Sets

Table V summarizes the dataset used in our evaluation. The first eight rows of the table correspond to attack campaigns carried out by a red team as part of the TC program. This set spans a period of 358 hours, and contains about 73 million events. The last row corresponds to benign data collected over a period of 3 to 5 days across four Linux servers in our research laboratory. Attack data sets were collected on Windows (W-1 and W-2), Linux (L-1 through L-3) and FreeBSD (F-1 through F-3) by three research teams that are also part of the TC program.

The "duration" column in Table V refers to the length of time for which audit data was emitted from a host. Note that this period covers both benign activities and attack related activities on a host. The next several columns provide a break down of audit log events into different types of operations. File open and close operations were not included in W-1 and W-2 data sets. Note that "read" and "write" columns include not only file reads/writes, but

TABLE V

Dataset for each campaign with duration, distribution of different system calls and total number of events.

Dataset	Duration (hh-mm-ss)	Open	Connect+ Accept	Read	Write	Clone+ Exec	Close+ Exit	Mmap / Loadlib	Others	Total Events	Scenario Graph
W-1	06:22:42	N/A	22.14%	44.70%	5.12%	3.73%	3.88%	17.40%	3.02%	100K	Figure 28
W-2	19:43:46	N/A	17.40%	47.63%	8.03%	3.28%	3.26%	15.22%	5.17%	401K	Figure 4
L-1	07:59:26	37%	0.11%	18.01%	1.15%	0.92%	38.76%	3.97%	0.07%	2.68M	Figure 25
L-2	79:06:39	39.58%	0.08%	12.19%	2%	0.83%	41.28%	3.79%	0.25%	38.5M	-
L-3	79:05:13	38.88%	0.04%	11.81%	2.35%	0.95%	40.98%	4.14%	0.84%	19.3M	Figure 29
F-1	08:17:30	9.46%	0.40%	24.65%	40.86%	2.10%	12.55%	9.08%	0.89%	701K	Figure 26
F-2	78:56:48	11.78%	0.42%	16.60%	44.52%	2.10%	15.04%	8.54%	1.01%	5.86M	Figure 27
F-3	79:04:54	11.31%	0.40%	19.46%	45.71%	1.64%	14.30%	6.16%	1.03%	5.68M	Figure 5
Benign	329:11:40	11.68%	0.71%	26.22%	30.03%	0.63%	15.42%	14.32%	0.99%	32.83M	N/A

also network reads and writes on Linux. However, on Windows, only file reads and writes were reported. Operations to load libraries were reported on Windows, but memory mapping operations weren't. On Linux and FreeBSD, there are no load operations, but most of the mmap calls are related to loading. So, the mmap count is a loose approximation of the number of loads on these two OSes. The "Others" column includes all the remaining audit operations, including `rename`, `link`, `rm`, `unlink`, `chmod`, `setuid`, and so on. The last column in the table identifies the scenario graph constructed by SLEUTH for each campaign. Due to space limitations, we have omitted scenario graphs for campaign L-2.

3.6.3 Engagement Setup

The attack scenarios in our evaluation are setup as follows. Five of the campaigns (i.e., W-2, L-2, L3, F-2, and F3) ran in parallel for 4 days, while the remaining three (W-1, L-1, and F-1) were run in parallel for 2 days. During each campaign, the red team carried out a series of attacks on the target hosts. The campaigns are aimed at achieving varying adversarial objectives, which include dropping and execution of an executable, gathering intelligence about a target host, backdoor injection, privilege escalation, and data exfiltration.

Being an adversarial engagement, we had no prior knowledge of the attacks planned by the red team. We were only told the broad range of attacker objectives described in the previous paragraph. It is worth noting that, while the red team was carrying out attacks on the target hosts, benign background activities were also being carried out on the hosts. These include activities such as browsing and downloading files, reading and writing emails, document processing, and so on. On average, *more than 99.9% of the events corresponded to* *benign activity.* Hence, SLEUTH had to automatically detect and reconstruct the attacks from a set of events including both benign and malicious activities.

We present our results in comparison with the ground truth data released by the red team. Before the release of ground truth data, we had to provide a report of our findings to the red team. The findings we report in this chapter match the findings we submitted to the red team. A summary of our detection and reconstruction results is provided in a tabular form in Table VII. Below, we first present reconstructed scenarios for selected datasets before proceeding to a discussion of these summary results.

3.6.4 Selected Reconstruction Results

Of the 8 attack scenarios successfully reconstructed by SLEUTH, we discuss campaigns W-2 (Windows) and F-3 (FreeBSD) in this section, while deferring the rest to Appendix A. To make it easier to follow the scenario graph, we provide a narrative that explains how the attack unfolded. This narrative requires manual interpretation of the graph, but the graph generation itself is automated. In these graphs, edge labels include the event name and a sequence number that indicates the global order in which that event was performed. Ovals, diamonds and rectangles represent processes, sockets and files, respectively.

Campaign W-2. Figure 4 shows the graph reconstructed by SLEUTH from Windows audit data. Although the actual attack campaign lasted half an hour, the host was running benign background activities for 20 hours. These background activities corresponded to more than 99.8% of the events in the corresponding audit log.

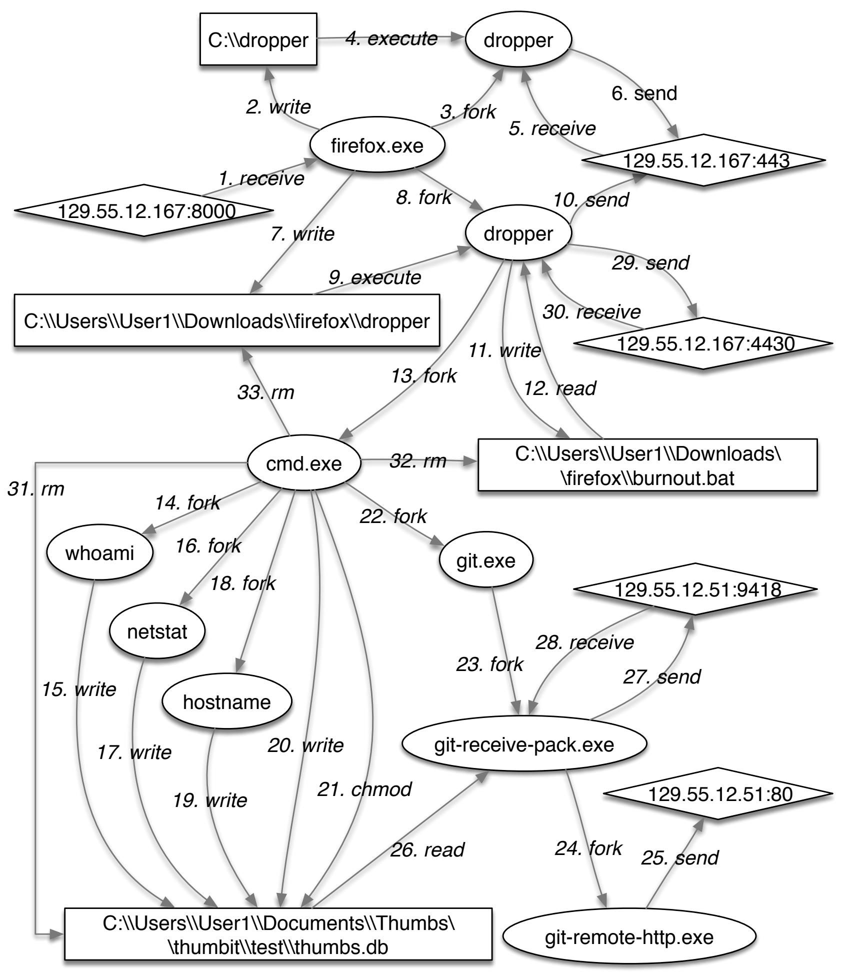


Figure 4. Scenario graph reconstructed from campaign W-2.

Entry: The initial entry point for the attack is `Firefox`, which is compromised on visiting the web server `129.55.12.167`.

Backdoor insertion: Once Firefox is compromised, a malicious program called `dropper` is downloaded and executed. Dropper seems to provide a remote interactive shell, connecting to ports 443 and then 4430 on the attack host, and executing received commands using `cmd.exe`.

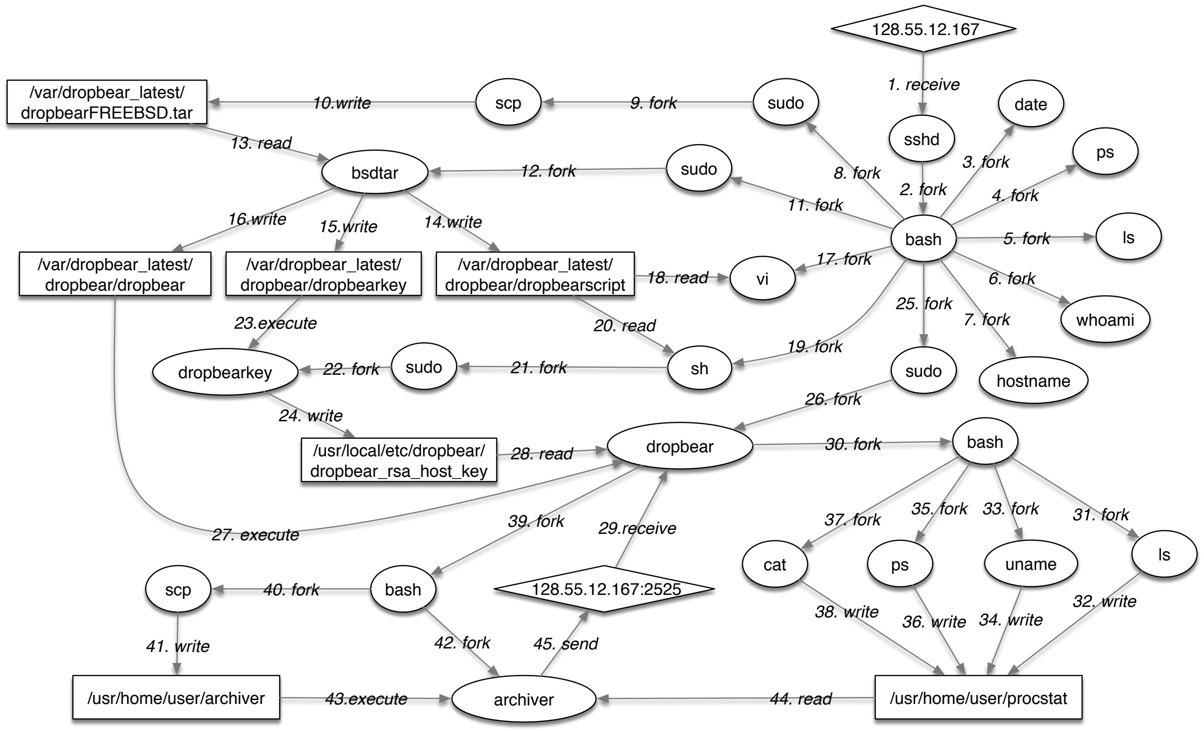


Figure 5. Scenario graph reconstructed from campaign F-3.

Intelligence gathering: Dropper then invokes `cmd.exe` multiple times, using it to perform various data gathering tasks. The programs `whoami`, `hostname` and `netstat` are being used as stand-ins for these data gathering applications. The collected data is written to `C:\Users\User1\-Documents\Thumbs\thumbit\test\thumbs.db`.

Data exfiltration: Then the collected intelligence is exfiltrated to 129.55.12.51:9418 using `git`.

Clean-up: Dropper downloads a batch file called `burnout.bat`. This file contains commands to clean up the attack footprint, which are executed by `cmd.exe` (see edges 11,12, 31-33).

Campaign F-3. (Figure 5). Under the command of an attacker who uses stolen ssh credentials, `sshd` forks a `bash` process. Note that though there is no direct evidence from the audit data about the stolen ssh credentials, because of the subsequent events (`scp`) from this shell, we conclude this as a sign of an attacker that uses stolen ssh credentials.

Next the attacker, invokes `scp`, which downloads a file into location `/var/dropbear_latest/-dropbearFREEBSD.tar`, which is then uncompressed. The file `dropbearscript` is next read and interpreted by `sh`. This action creates the process `dropbearkey`, which writes to `/usr/ local/etc/-dropbear/dropbear_ecdsa_host_key` and `/usr/local/etc/dropbear/dropbear_rsa_host_key`. Next, another `sudo` process created by `bash` starts another `dropbear` process which reads these two keys for future use (presumably to assist in connecting to a remote host).

Dropbear next starts a shell process, which executes a series of commands `ls`, `bash`, `uname`, `ps`, all of which write to a file `/usr/home/user/procstats`.

Finally, `dropbear` starts a `bash` process, which uses `scp` to download a file called `/usr/home/-user/archiver`, and executes that file. The resulting process, called `archiver`, reads the file `/usr/home/user/procstats`, which contains the data output earlier, and exfiltrates this information to `128.55.12.167:2525`.

Summary. The above two graphs were constructed automatically by SLEUTH from audit data. They demonstrate how SLEUTH enables an analyst to obtain compact yet complete attack scenarios from hours of audit data. SLEUTH is able to hone in on the attack activity, even when it is hidden among benign data that is at least three orders of magnitude larger.

3.6.5 Overall Effectiveness

To assess the effectiveness of Sleuth in capturing essential stages of an APT, in Table VI, we correlate pieces of attack scenarios constructed by Sleuth with APT stages documented in postmortem reports of notable APT campaigns (e.g., the MANDIANT [13] report). In 7 of the 8 attack scenarios, Sleuth uncovered the drop&load activity. In all the scenarios, Sleuth captured concrete evidence of data exfiltration, a key stage in an APT campaign. In 7 of the scenarios, commands used by the attacker to gather information about the target host were captured by Sleuth.

Another distinctive aspect of an APT is the injection of backdoors to targets and their use for C&C and data exfiltration. In this regard, 6 of the 8 scenarios reconstructed by Sleuth involve backdoor injection. Cleaning the attack footprint is a common element of an APT campaign. In our experiments, in 5 of the 8 scenarios, Sleuth uncovered attack cleanup activities, e.g., removing dropped executables and data files created during the attack.

TABLE VI

Sleuth results with respect to a typical APT campaign.

Dataset	Drop & Load	Intelligence Gathering	Backdoor Insertion	Privilege Escalation	Data Exfiltration	Cleanup
W-1	✓	✓			✓	✓
W-2	✓	✓	✓		✓	✓
L-1	✓	✓	✓		✓	✓
L-2	✓	✓	✓	✓	✓	✓
L-3	✓	✓	✓	✓	✓	✓
F-1			✓		✓	
F-2	✓	✓	✓		✓	
F-3	✓	✓			✓	

Table VII shows another way of breaking down the attack scenario reconstruction results, counting the number of key files, network connections, and programs involved in the attack. Specifically, we count the number of attack entry entities (including the entry points and the processes that communicate with those entry points), attack-related program executions, key files that were generated and used during the campaign, and the number of exit points used for exfiltration (e.g., network sockets). This data was compared with the ground truth, which was made available to us after we obtained the results. The last two columns show the incorrectly reported and missed entities, respectively.

The two missed entities were the result of the fact that we had not spent any effort in cataloging sensitive data files and device files. As a result, these entities were filtered out during the forward analysis and simplification steps. Once we marked the two files correctly, they were no longer filtered out, and we were able to identify all of the key entities.

In addition to the missed entities shown in Table VII, the red team reported that we missed a few other attacks and entities. Some of these were in data sets we did not examine. In

TABLE VII

Attack scenario reconstruction summary.

Dataset	Entry Entities	Programs Executed	Key Files	Exit Points	Correctly Identified Entities	Incorrectly Identified Entities	Missed Entities
W-1	2	8	7	3	20	0	0
W-2	2	8	4	4	18	0	0
L-1	2	10	7	2	20	0	1
L-2	2	20	11	4	37	0	0
L-3	1	6	6	5	18	0	0
F-1	4	13	9	2	13	0	1
F-2	2	10	7	3	22	0	0
F-3	4	14	7	1	26	0	0
Total	**19**	**89**	**58**	**24**	**174**	**0**	**2**

particular, campaign W-2 was run multiple times, and we examined the data set from only one instance of it. Also, there was a third attack campaign W-3 on Windows, but the team producing Windows data sets had difficulties during W-3 that caused the attack activities not to be recorded, so that data set is omitted from the results in Table VII. Similarly, the team responsible for producing Linux data sets had some issues during campaign L-3 that caused some attack activities not to be recorded. To account for this, Table VII counts only the subset of key entities whose names are present in the L-3 data set given to us.

According to the ground truth provided by the red team, we incorrectly identified 21 entities in F-1 that were not part of an attack. Subsequent investigation showed that the auditing system had not been shutdown at the end of the F-1 campaign, and all of these false positives correspond to testing/administration steps carried out after the end of the engagement, when the auditing system should not have been running.

3.6.6 False Alarms in a Benign Environment

In order to study SLEUTH's performance in a benign environment, we collected audit data from four Ubuntu Linux servers over a period of 3 to 5 days. One of these is a mail server,

TABLE VIII

False alarms in a benign environment with software upgrades and updates. No alerts were triggered during this period.

Dataset	Log Size on Disk	# of Events	Duration (hh:mm:ss)	Packages Updated	Binary Files Written
Server 1	1.1G	2.17M	00:13:06	110	1.8K
Server 2	2.7G	4.67M	105:08:22	4	4.2K
Server 3	12G	20.9M	104:36:43	4	4.3K
Server 4	3.2G	5.09M	119:13:29	4	4.3K

another is a web server, and a third is an NFS/SSH/SVN server. Our focus was on software updates and upgrades during this period, since these updates can download code from the network, thereby raising the possibility of untrusted code execution alarms. There were four security updates (including kernel updates) performed over this period. In addition, on a fourth server, we collected data when a software upgrade was performed, resulting in changes to 110 packages. Several thousand binary and script files were updated during this period, and the audit logs contained over 30M events. All of this information is summarized in Table VIII.

As noted before, policies should be configured to permit software updates and upgrades using standard means approved in an enterprise. For Ubuntu Linux, we had one policy rule for this: when `dpkg` was executed by `apt`-commands, or by `unattended-upgrades`, the process is not downgraded even when reading from files with untrusted labels. This is because both `apt` and `unattended-upgrades` verify and authenticate the hash on the downloaded packages, and only after these verifications do they invoke `dpkg` to extract the contents and write to various directories containing binaries and libraries. Because of this policy, all of the 10K+ files downloaded were marked benign. As a result of this, no alarms were generated from their execution by SLEUTH.

3.6.7 Runtime and Memory Use

Table IX shows the runtime and memory used by SLEUTH for analyzing various scenarios. The measurements were made on a Ubuntu 16.04 server with 2.8GHz AMD Opteron 62xx processor and 48GB main memory. Only a single core of a single processor was used. The first column shows the campaign name, while the second shows the total duration of the data set.

The third column shows the memory used for the dependence graph. As described in Section 3.2, we have designed a main memory representation that is very compact. This compact representation enables Sleuth to store data spanning very long periods of time. As an example, consider campaign L-2, whose data were the most dense. Sleuth used approximately 329MB to store 38.5M events spanning about 3.5 days. Across all data sets, Sleuth needed about 8 bytes of memory per event on the larger data sets, and about 20 bytes per event on the smaller data sets.

The fourth column shows the total run time, including the times for consuming the dataset, constructing the dependence graph, detecting attacks, and reconstructing the scenario. We note that this time was measured after the engagement when all the data sets were available. During the engagement, Sleuth was consuming these data as they were being produced. Although the data typically covers a duration of several hours to a few days, the analysis itself is very fast, taking just seconds to a couple of minutes. Because of our use of tags, most information

TABLE IX

Memory use and runtime for scenario reconstruction.

Dataset	Duration (hh:mm:ss)	Memory Usage	Runtime	
			Time	Speed-up
W-1	06:22:42	3 MB	1.19 s	19.3 K
W-2	19:43:46	10 MB	2.13 s	33.3 K
W-Mean		6.5 MB		**26.3 K**
L-1	07:59:26	26 MB	8.71 s	3.3 K
L-2	79:06:39	329 MB	114.14s	2.5 K
L-3	79:05:13	175 MB	74.14 s	3.9 K
L-Mean		177 MB		**3.2 K**
F-1	08:17:30	8 MB	1.86 s	16 K
F-2	78:56:48	84 MB	14.02 s	20.2 K
F-3	79:04:54	95 MB	15.75 s	18.1 K
F-Mean		62.3 MB		**18.1 K**

needed for the analysis is locally available. This is the principal reason for the performance we achieve.

The "speed-up" column illustrates the performance benefits of SLEUTH. It can be thought of as the number of simultaneous data streams that can be handled by SLEUTH, if CPU use was the only constraint.

In summary, SLEUTH is able to consume and analyze audit COTS data from several OSes in real time while having a small memory footprint.

3.6.8 Benefit of split tags for code and data

As described earlier, we maintain two trustworthiness tags for each subject, one corresponding to its code, and another corresponding to its data. By prioritizing detection and forward analysis on code trustworthiness, we cut down vast numbers of alarms, while greatly decreasing the size of forward analysis output.

TABLE X

Reduction in (false) alarms by maintaining separate code and data trustworthiness tags. The average reduction shows the average factor of reduction we get for alarms generation when using split trustworthiness tag over single trustworthiness tag.

Dataset	Untrusted execution		Modification by low code t-tag subject		Preparation of untrusted data for execution		Confidential data leak	
	Single t-tag	Split t-tags	Single t-tag	Split t-tags	Single t-tags	Split t-tags	Single t-tag	Split t-tags
W-1	21	3	1.2 K	3	0	0	6.1 K	11
W-2	44	2	3.7 K	108	0	0	20.2 K	18
L-1	60	2	53	5	1	1	19	6
L-2	1.5 K	5	19.5 K	1	280	8	122 K	159
L-3	695	5	26.1 K	2	270	0	62.1 K	5.3 K
Average Reduction	**45.39x**		**517x**		**6.24x**		**112x**	

Table X shows the difference between the number of alarms generated by our four detection policies with single trustworthiness tag and with the split trustworthiness (code and integrity) tags. Note that the split reduces the alarms by a factor of 100 to over 1000 in some cases.

Table XI shows the improvement achieved in forward analysis as a result of this split. In particular, the increased selectivity reported in column 5 of this table comes from splitting the tag. Note that often, there is a 100x to 1000x reduction in the size of the graph.

3.6.9 Analysis Selectivity

Table XI shows the data reduction pipeline of the analyses in SLEUTH. The second column shows the number of original events in each campaign. These events include all the events in the system (benign and malicious) over several days with an overwhelming majority having a benign nature, unrelated to the attack.

The third column shows the final number of events that go into the attack scenario graph.

TABLE XI

Comparison of selectivity achieved using forward analysis with single trustworthiness tags, forward analysis with split code and data trustworthiness tags, and finally simplifications.

Dataset	Initial # of Events	Final # of Events	Reduction Factor			
			Single t-tag	Split t-tag	SLEUTH Simplif.	Total
W-1	100 K	51	4.4x	1394x	1.4x	1951x
W-2	401 K	28	3.6x	552x	26x	14352x
L-1	2.68 M	36	8.9x	15931x	4.7x	74875x
L-2	38.5 M	130	7.3x	2971x	100x	297100x
L-3	19.3 M	45	7.6x	1208x	356x	430048x
F-1	701 K	45	2.3x	376x	41x	15416x
F-2	5.86 M	39	1.9x	689x	218x	150202x
F-3	5.68 M	45	6.7x	740x	170x	125800x
Average Reduction			**4.68x**	**1305x**	**41.8x**	**54517x**

The fourth column shows the reduction factor when a naive forward analysis with single trustworthiness tag (single t-tag) is used from the entry points identified by our backward analysis. Note that the graph size is very large in most cases. The fifth column shows the reduction factor using the forward analysis of SLEUTH— which is based on split (code and data) trustworthiness tags. As can be seen from the table, SLEUTH achieved two to three orders of magnitude reduction with respect to single t-tag based analysis.

The output of forward analysis is then fed into the simplification engine. The sixth column shows the reduction factor achieved by the simplifications over the output of our forward analysis. The last column shows the overall reduction we get over original events using split (code and data) trustworthiness tags and performing the simplification.

Overall, the combined effect of all of these steps is very substantial: data sets consisting of tens of millions of edges are reduced into graphs with perhaps a hundred edges, representing five orders of magnitude reduction in the case of L-2 and L-3 data sets, and four orders of magnitude reduction on other data.

3.7 Summary

We presented an approach and a system called SLEUTH for real-time detection of attacks and attack reconstruction from COTS audit logs. SLEUTH uses a main memory graph data model and a rich tag-based policy framework that make its analysis both efficient and precise. We evaluated SLEUTH on large datasets from 3 major OSes under attack by an independent red team, efficiently reconstructing all the attacks with very few errors.

CHAPTER 4

HOLMES: REAL-TIME APT DETECTION THROUGH CORRELATION OF SUSPICIOUS INFORMATION FLOWS

This chapter includes excerpts and figures from material that is published in [2].

4.1 Introduction

The main problem tackled in this chapter is to detect an ongoing APT campaign (that consists of many disparate steps across many hosts over a long period of time) in real-time and provide a high-level explanation of the attack scenario to an analyst, based on host logs and IPS alerts from the enterprise. There are three main aspects to this problem, and they are as follows:

- Alert generation: Starting from low-level event traces from hosts, we must generate alerts in an efficient manner. How do we generate alerts that attempt to factor any significant steps the attacker might be taking? Additionally, care must be taken to ensure that we do not generate a large volume of noisy alerts.
- Alert correlation: The challenge here is to combine these alerts from multiple activities of the attacker into a reliable signal that indicates the presence of an ongoing APT campaign.
- Attack scenario presentation: Indicators of an ongoing APT campaign needs to be communicated to a human being (a cyber-analyst). To be effective, this communication must

be intuitive and needs to summarize the attack at a high level such that the analyst quickly realizes the scope and magnitude of the campaign.

We present a system called Holmes in this chapter that addresses all the above aspects. Holmes begins with host audit data and produces a detection signal that maps out the stages of an ongoing APT campaign. At a high level, Holmes makes *novel use of the APT kill-chain* as the pivotal reference in addressing the technical challenges involved in the above three aspects of APT detection. We describe our key ideas and their significance below, with a detailed technical description appearing in Section 4.3.

First, Holmes aims to map the activities found in host logs as well as any alerts found in the enterprise directly to the kill chain. This design choice allows Holmes to generate alerts that are semantically close to the activity steps ("Tactics, Techniques and Procedures" (TTPs)) of APT actors. By doing so, Holmes elevates the alert generation process to work at the level of the steps of an attack campaign, than about how they manifest in low-level audit logs. Thus, we solve an important challenge in generating alerts of significance. In our experiments, we have found that a five-day collection of audit logs contains around 3M low-level events, while Holmes only extracts 86 suspicious activity steps from them.

A second important idea in Holmes is to *use the information flow between low-level entities (files, processes, etc.) in the system as the basis for alert correlation.* To see this, note that the internal reconnaissance step in the kill-chain depends on a successful initial compromise and establishment of a foothold. In particular, the reconnaissance step is typically launched using the command and control agent (process) installed by the attacker during foothold establishment, thus exhibiting a flow between the processes involved in the two phases. Moreover, reconnaissance often involves running malware (files) downloaded during the foothold establishment phase, illustrating a file-to-process flow. Similarly, a successful lateral movement phase, as well as the exfiltration phase, uses data gathered by the reconnaissance phase. Thus, by detecting low-level events associated with APT steps and linking them using information flow, it is possible to construct the emerging kill-chain used by an APT actor.

A third main contribution in HOLMES is the development of a high-level scenario graph (HSG). The nodes of the HSG correspond to TTPs, and the edges represent information flows between entities involved in the TTPs. The HSG provides the basis for detecting APTs with high confidence. For this purpose, we develop several new ideas. First is the concept of an *ancestral cover* in an HSG. We show how this concept can help to assess the strength of dependencies between HSG nodes. Weak dependencies can then be pruned away to eliminate many false alarms. Second, we develop *noise reduction* techniques that further de-emphasize dependencies that are known to be associated with benign activities. Third, we develop ranking and prioritization techniques to prune away most nodes and edges unrelated to the APT campaign. These steps are described in detail in Sections 4.4.4, 4.4.5, and 4.4.6. Using these techniques, we demonstrate that HOLMES is able to make a clear distinction between attack and benign scenarios.

Finally, the HSG provides a very compact, visual summary of the campaign at any moment, thus making an important contribution for attack comprehension. For instance, starting from a dataset of 10M audit records, we are able to summarize a high-level attack campaign using a graph of just 16 nodes. A cyber-analyst can use the presented HSG to quickly infer the big picture of the attack (scope and magnitude) with relative ease.

Evaluation. We evaluated HOLMES on data generated by TC program. The advantage of using system audit data is that it is a reliable source of information and is free of unauthorized tamper (under a threat model of non-compromised kernel). Evaluation of HOLMES on nine real-life APT attack scenarios, as well as running it as a real-time intrusion detection tool in a live experiment spanning for two weeks, show that HOLMES is able to clearly distinguish between attack and benign scenarios and can discover cyber-attacks with high precision and recall (Section 4.6).

4.2 A Running Example

In this section, we present a running example used through the chapter to illustrate our approach. This example represents an attack carried out by a red-team as part of the TC program. In this attack, a vulnerable *Nginx* web server runs on a *FreeBSD* system. Its operations (system calls) are captured in the system audit log. From this audit data, we construct a *provenance graph*, a fragment of which is shown in Figure 6. Nodes in this graph represent system entities such as processes (represented as rectangles), files (ovals), network connections (diamonds), memory objects (pentagons), and users (stars). Edges correspond to system calls and are oriented in the direction of information flow and/or causality. Note that our provenance graph has been rendered acyclic using the (optimized) node versioning technique described in Reference [36].

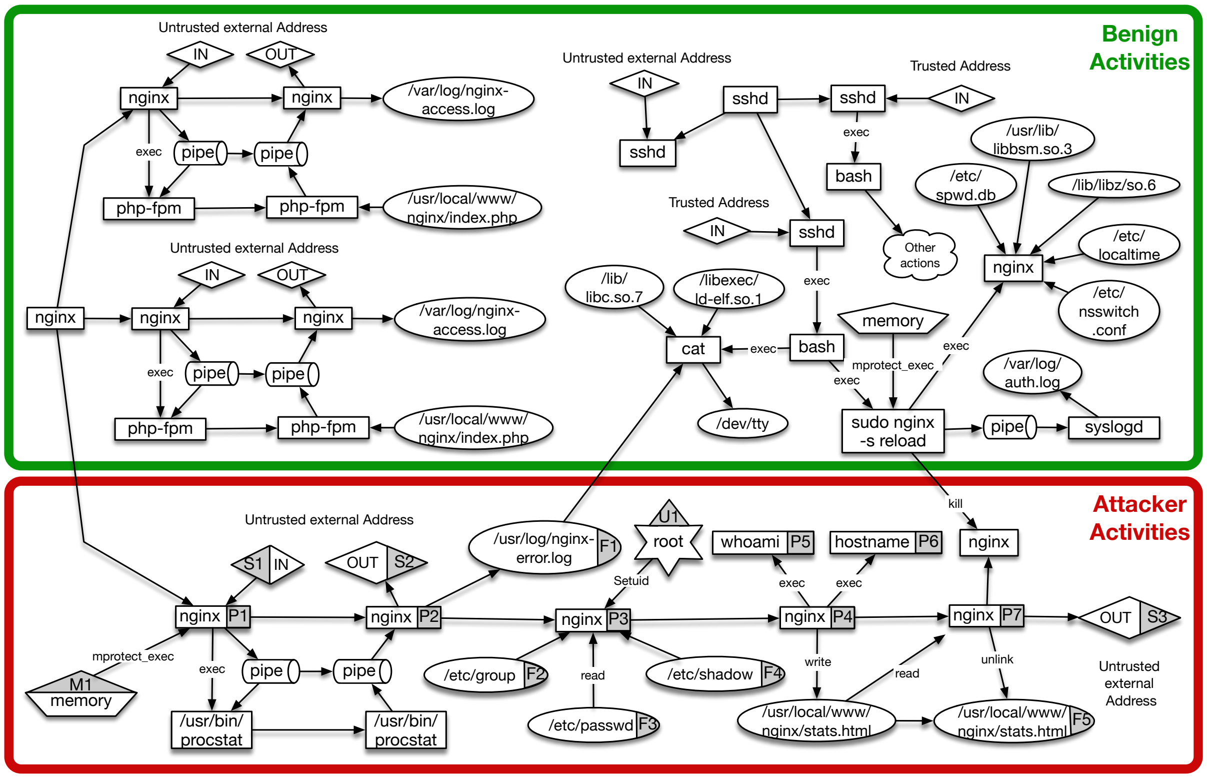


Figure 6. Provenance Graph of the Running Example.

The goal of the attacker is to exfiltrate sensitive information from the system. The attacker's activities are depicted at the bottom of Figure 6, and consist of the following steps:

- *Initial Compromise.* The attacker sends a malicious payload on the socket (S1) listening on port 80. As a result, *Nginx* makes some part of its memory region (M1) executable. Next, the attacker gains control over the *Nginx* process by using a reflective self-loading exploit.

- *C&C Communications.* The compromised *Nginx* process makes a connection (S2) to the C&C server to receive commands from the attacker.
- *Privilege Escalation.* The attacker exploits an existing vulnerability to escalate the privilege of *Nginx* to root (U1).
- *Internal Reconnaissance.* Next, the attacker issues commands such as *whoami* (P5) and *hostname* (P6). These commands were used by the red team to simulate access to confidential/proprietary data. The attacker also reads usernames and password hashes (F2, F3, F4) and writes all this information to a temporary file.
- *Exfiltration.* Next, the attacker transfers the file containing the gathered information to her/his machine (S3).
- *Cleanup.* In the last step of the attack, the attacker removes the temporary file (F5) to clean up any attack remnants.

This example illustrates many key challenges described below:

Stealthy Attacks. This attack leaves a minimal footprint on the system. The first step of the attack, the initial compromise of the *Nginx* server, is executed in main memory and does not leave any visible traces such as downloaded files. Moreover, the payload runs within the existing `Nginx` process. It is very challenging to detect such stealthy attacks, where attacker activities blend in seamlessly with normal system operation.

Needle in a haystack. Even a single host can generate tens of millions of events per day. All but a very tiny fraction of these — typically much less than 0.01% — correspond to benign activities. (The top portion of Figure 6 shows a small subset of benign activities in the audit log.) It is difficult to detect such rare events without a high rate of false alarms. More importantly, it is very challenging to filter out these benign events from the attack summaries presented to analysts.

Real-time detection. We envision Holmes to be used in conjunction with a *cyber-response* system, so it is necessary to detect and summarize an ongoing campaign in a matter of seconds. Real-time detection poses additional challenges and constraints for the techniques used in Holmes.

To overcome these challenges, note that, despite blending seamlessly into benign background activity, two factors stand out regarding the attack. First, the attack steps achieve capabilities corresponding to some of the APT stages. Second, the attack activities are connected via information flows. In the next section, we describe the Holmes approach based on these two key observations.

4.3 Approach Overview

The central insight behind our approach is that even though the concrete attack steps may vary widely among different APTs, the high-level APT behavior often conforms to the same kill-chain introduced in Section 4 (Figure 1). Our analysis of hundreds of APT reports from [5] suggests that most APTs consist of a subset, if not all, of those steps. More importantly, we make the observation that these steps need to be *causally connected*, and this connectedness is a major indication that an attack is unfolding.

Note that the concrete manifestation of each APT step may vary, e.g., an initial compromise may be executed as a drive-by-download or as a spear-phishing attack with a malicious file that is executed by a user. Regardless, the APT steps themselves represent a high-level abstraction of the attacker's intentions, and hence they must manifest themselves even if the operational tactics used by attackers vary across APTs. Moreover, information flow or causal relations must necessarily exist between them since the APT steps are logically dependent on each other, e.g., exfiltration is dependent on internal reconnaissance to gather sensitive data.

The research question, therefore, is whether we can base our detection on

- an APT's most essential high-level behavioral steps, and
- the information flow dependencies between these steps.

A major challenge in answering this question is the large semantic gap between low-level audit data and the very high-level kill-chain view of attacker's goals, intentions, and capabilities.

Bridging the Semantic Gap. To bridge the semantic gap between low-level system-call view and the high-level kill-chain view, we build an intermediate layer as shown in Figure 7. The mapping to this intermediate layer is based on MITRE's ATT&CK framework [10], which describes close to 200 behavioral patterns defined as *Tactics, Techniques, and Procedures (TTPs)* observed in the wild.

Each TTP defines one possible way to realize a particular high-level capability. For instance, the capability of *persistence* in a compromised Linux system can be achieved using 11 distinct TTPs, each of which represents a possible sequence of lower level actions in the ATT&CK framework, e.g., installation of a rootkit, modification of boot scripts, and so on. These lower

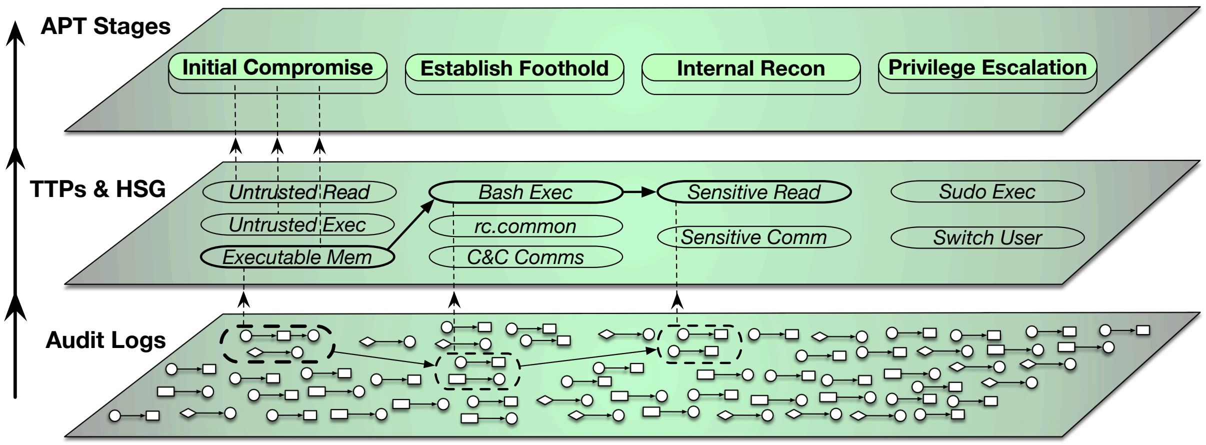


Figure 7. Holmes Approach: From Audit Records to High-Level APT Stages

level actions are closer to the level of abstraction of audit logs, so it is possible to describe TTPs in terms of nodes and edges in the provenance graph.

Technical challenges. The main technical challenges in realizing the approach summarized in Figure 7 are:

- *efficient matching* of low-level event streams to TTPs,
- *detecting correlation* between attack steps, and
- *reducing* false positives.

We solve these challenges through several design innovations. For efficient matching, we use a representation of the audit logs as a directed provenance graph (Section 4.4) in main memory, which allows for efficient matching. This graph also encodes the information flow dependencies that exist between system entities (such as processes and files). TTPs are specified as patterns that leverage these dependencies. For instance, in order to match a *maintain persistence* TTP, an information flow dependency must exist from a process matching an *initial compromise* TTP to the *maintain persistence* TTP.

For detecting correlations between attack steps, we build a *High-level Scenario Graph (HSG)* as an abstraction over the provenance graph. Each node in the HSG represents a matched TTP, while the edges represent information flow and causality dependencies among those matched TTPs. An HSG is illustrated in the middle layer of Figure 7 by nodes and edges in boldface. (We refer the reader to Figure 8 for the HSG of the running example.) To determine the edges among nodes in the HSG, use the *prerequisite-consequence* patterns of among the TTPs and the APT stages.

To reduce the number of false positives (i.e., HSGs that do not represent attacks), we use a combination of: (a) learning benign patterns that may produce false positive TTPs and, (b) heuristics that assign weights to nodes and paths in the graph based on their severity, so that the HSGs can be ranked, and the highest-ranked HSGs presented to the analyst.

In summary, the high-level phases of an APT are operationalized using a common suite of tactics that can be observed from audit data. These observations provide evidence that some malicious activity may be unfolding. The job of Holmes, then, is to collect pieces of evidence and infer the correlations among them and use these correlations to map out the overall attack campaign.

4.4 System Design

Like most previous works [25, 38–40] that rely on OS audit data, we consider the OS kernel and the auditing engine as part of the trusted computing base (TCB). In other words, attacks on the OS kernel, the auditing system and the logs produced by it are outside the scope of our threat model. We also assume that the system is benign at the outset, so the initial attack must originate external to the enterprise, using means such as remote network access, removable storage, etc.

4.4.1 Data Collection and Representation

HOLMES relies on the provenance graphs constructed from audit logs retrieved from multiple hosts that may run different operating systems (OSes). [1] As mentioned in Chapter 3, we use a highly compact provenance graph representation that, on average, requires less than 5 bytes of main memory per event in the audit log. In addition the entities of our provenance graph are *versioned.* A new version of a node is created before adding an incoming edge if this edge changes the existing dependencies (i.e., the set of ancestor nodes) of the node. Versioning enables optimizations that can prune away a large fraction of events in the audit log without changing the results of forensic analysis [36]. Moreover, this versioned graph is acyclic, which can simplify many graph algorithms. This representation enables real-time consumption of events and graph construction over prolonged periods of time. It is on this provenance graph that our analysis queries for behavior that matches our TTP specifications.

[1] The design of HOLMES makes it possible to take additional inputs such as events and alerts from a variety of IDS/IPS, but we do not discuss this aspect of the system further.

4.4.2 TTP Specification

TTP specifications provide the mapping between low-level audit events and high-level APT steps. Therefore, they are a central component of our approach. In this subsection, we describe three key choices in the TTP design that enable efficient and precise attack detection.

Recall that in our design, TTPs represent a layer of intermediate abstraction between concrete audit logs and high-level APT steps. Specifically, we rely on two main techniques to lift audit log data to this intermediate layer: (a) an OS-neutral representation of security-relevant events in the form of the *provenance graph* and (b) use of *information flow* dependencies between entities involved in the TTPs. Taken together, these techniques enable high-level specifications of malicious behavior that are largely independent of many TTP details such as the specific system calls used, names of malware, intermediate files that were created and the programs used to create them, etc. In this regard, our information flow based TTP specification approach is more general than the use of *misuse specifications* [41, 42] from the IDS literature. Use of information flow dependencies is crucial in the detection of stealthy APTs that hide their activities by using benign system processes to carry out their goals.

In addition to specifying the steps of a TTP, we need to capture its prerequisites. Prerequisites not only help reduce false positives but also help in understanding the role of a TTP in the larger context of an APT campaign. In our TTP specifications, prerequisites take the form of causal relationships and information flows between APT stages.

Finally, TTP matching needs to be efficient, and must not require expensive techniques such as backtracking. We find that most TTPs can be modeled in our framework using a single event, with additional preconditions on the subjects and objects involved.

An example of a TTP rule specification is shown in Table XII, with additional rules appearing in Section 4.5. In Table XII, the first column represents the APT stage, and the second column represents the associated TTP name and the entities involved in the TTP. The third column specifies the event family associated with the TTP. For ease of illustration, some of the specific events included in this family are shown in the fourth column, but note that they are not part of a TTP rule. (Event classes are defined once, and reused across all TTP rules.)

TABLE XII

Example TTPs. In the Severity column, L=Low, M=Moderate, H=High, C=Critical. Entity types are shown by the characters: P=Process, F=File, S=Socket, M=Memory, U=User.

APT Stage	**TTP**	**Event Family**	**Events**	**Severity**	**Prerequisites**
$Initial\_Compromise(P)$	$Untrusted\_Read(S, P)$	READ	FileIoRead (Windows), read/pread/readv/preadv (Linux,BSD)	L	$S.ip \notin \{\text{Trusted\_IP\_Addresses}\}$
	$Make\_Mem\_Exec(P, M)$	MPROTECT	VirtualAlloc (Windows), mprotect (Linux,BSD)	M	$\$\text{PROT\_EXEC}\$ \in M.flags \wedge \exists\, Untrusted\_Read(?, P') : path\_factor(P', P) <= path\_thres$
$Establish\_Foothold(P)$	$Shell\_Exec(F, P)$	EXEC	ProcessStart (Windows), execve/fexecve (Linux,BSD)	M	$F.path \in \{\text{Command\_Line\_Utilities}\} \wedge \exists\, Initial\_Compromise(P') : path\_factor(P', P) <= path\_thres$

The fifth column represents a severity level associated with each TTP. We use this severity level to rank alarms raised by our system, prioritizing the most severe alarms. Our current assignment of the severity levels is based on the Common Attack Pattern Enumeration and Classification (CAPEC) list defined by US-CERT and DHS with the collaboration of MITRE [43] but can be tailored to suit the needs of a particular enterprise. We also provide another customization mechanism, whereby each severity level can be mapped to an analyst-specified weight that reflects the relative importance of different APT stages in a deployment context.

The last column specifies the prerequisites for the TTP rule to match. The prerequisites can specify conditions on the parameters of the TTP being matched, e.g., the socket parameter S for the $Untrusted\_Read$ TTP on the first row. Prerequisites can also contain conditions on previously matched TTPs and their parameters. For instance, the prerequisite column of the $Make\_Mem\_Exec(P, M)$ TTP contains a condition $\exists\, Untrusted\_Read(?, P')$. This prerequisite is satisfied only if an $Untrusted\_Read$ TTP has been matched for a process P' earlier, and if the processes involved in the two TTPs have a $path\_factor$ (defined below) less than a specified threshold.

Prerequisites can capture relations between the entities involved in two TTPs, such as the parent-child relation on processes, or information flow between files. They can also capture the condition that two TTPs share a common parent. Using prerequisites, we are able to prune many false positives, i.e., benign activity resembling a TTP.

4.4.3 HSG Construction

Figure 8 illustrates an HSG for the running example. The nodes of this graph represent matched TTPs and are depicted by ovals in the figure. Inside each oval, we represent the matched provenance graph entities in grey. For illustration purposes, we have also included the name of the TTP, the APT stage to which each TTP belongs, and the severity level (Low, Medium or High) of each TTP. The edges of the graph represent the prerequisites between different TTPs. The dotted lines that complete a path between two entities represent the prerequisite conditions. For instance, the $Make\_Mem\_Exec$ TTP has, as a prerequisite, an $Untrusted\_Read$ TTP, represented by the edge between the two nodes.

The construction of the HSG is primarily driven by the *prerequisites:* A TTP is matched and added to the HSG if all its prerequisites are satisfied. This factor reduces the number of TTPs in the HSG at any time, making it possible to carry out sophisticated analyses without impacting real-time performance.

4.4.4 Avoiding Spurious Dependencies

By *spurious dependencies*, we refer to uninteresting and/or irrelevant dependencies on the attacker's activities. For instance, in Figure 6, the process `nginx` (P2) writes to the file `/usr/log/nginx-error.log`, and the `cat` process later reads that file. However, even though there is a dependency between `cat` and the log file, `cat` is unrelated to the attack and is invoked independently through `ssh`. More generally, consider any process that consumes secondary artifacts produced by the attack activity, e.g., a log rotation system that copies a log file containing some fraction of entries produced by an attacker's process. Such processes, although they represent benign background activity, will be flagged in the provenance graph as having a dependence on the attacker's processes. If these spurious dependencies aren't promptly pruned, there can be a dependence explosion that can enormously increase the size of HSGs. As a result, the final result presented to the analyst may be full of benign activities, which can cause

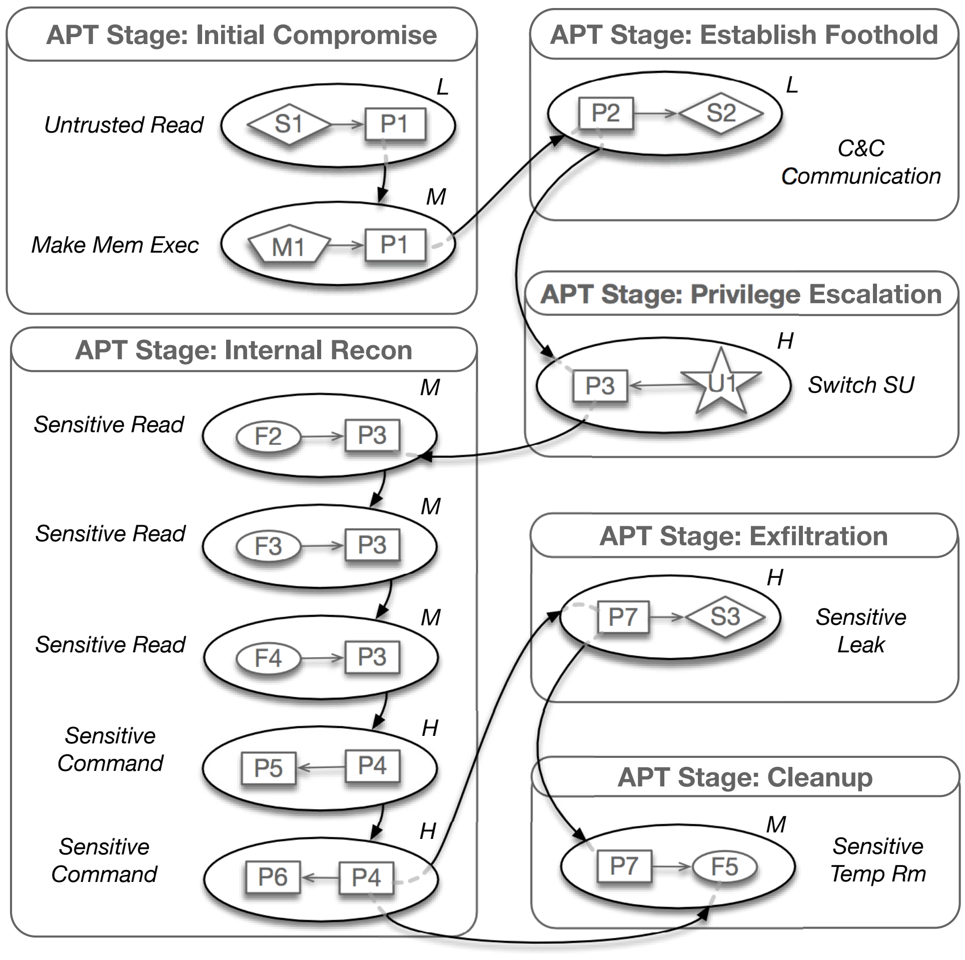


Figure 8. High-Level Scenario Graph for the Running Example.

the analyst to miss key attack steps embedded in a large graph. For this reason, we prioritize *stronger* dependencies over *weaker* ones, pruning away the latter as much as possible.

Intuitively, we can say that a process P_d has a *strong* dependency on a process P_a if P_d is a descendant process of P_a. Similarly, a file or a socket has a *strong* dependency on a process P_a if P_a or its descendant processes write to this file/socket. More generally, consider two entities and a path between them in the *provenance graph* that indicates an information between them. Determining if this flow represents a *strong* or *weak* information flow is equivalent to determining if the entities in the flow share *compromised* ancestors. If they share compromised ancestors, they are part of the attacker's activities, and there is a *strong* dependency among them, which must be prioritized. Otherwise, we consider the dependency to be weak and deemphasize it in our analysis.

To generalize the above discussion to a case where there may be multiple compromised processes, we introduce the following notion of an *ancestral cover* $AC(f)$ of all processes on an information flow path f:

$$\forall p \in f\ \exists a \in AC(f)\ \ a = p \text{ or } a \text{ is an ancestor of } p$$

Note that non-process nodes in f don't affect the above definition. A *minimum ancestral cover*, $AC_{min}(f)$ is an ancestral cover of minimum size. Intuitively, $AC_{min}(f)$ represents the minimum number of ancestors that an attacker must compromise (i.e., the number of exploits) to have full control of the information flow path f. For instance, consider again the flow from the nginx process, which is under the control of the attacker, to the cat process. Since these two processes share no common ancestors, the minimum ancestral cover for the path among them has a size that is equal to 2. Therefore, to control the cat process, an attacker would have to develop an additional exploit for cat. This requires the attacker to first find a vulnerability in cat, then create a corresponding exploit, and finally, write this exploit into the log file. By preferring an ancestral cover of size 1, we capture the fact that such an attack involving cat is a lot less likely than one where the attack activities are executed by nginx and its descendants.

We can now define the notion of $path\_factor(N_1, N_2)$ mentioned earlier in the discussion of TTPs. Intuitively, it captures the extent of the attacker's control over the flow from N_1 to N_2. Based on the above discussion of using minimum ancestral covers as a measure of dependency strength, we define $path\_factor$ as follows. Consider all of the information flow paths $f_1, ..., f_n$ from N_1 to N_2, and let m_i be the minimum ancestral cover size for f_i. Then, $path\_factor(N_1, N_2)$ is simply the minimum value among $m_1, \ldots, m_n$.

Note that if process N_2 is a child of N_1, then there is a path with just a single edge between N_1 to N_2. The size of minimum ancestral cover for this path is 1 since N_1 is an ancestor of N_2. In contrast, the (sole) path from nginx to cat has a minimum ancestral cover of size 2, so $path\_factor(\texttt{nginx}, \texttt{cat}) = 2$.

We describe an efficient computation of $path\_factor$ in Section 4.5. In our experience, the use of $path\_factor$ greatly mitigated dependency explosions by prioritizing attacker-influenced flows.

4.4.5 Noise Reduction

One of the challenges in the analysis of audit logs for attack detection and forensics is the presence of noise, i.e., benign events matching TTP rules. Long-living processes such as browsers, web servers, and SSH daemons trigger TTP matches from time to time. To cut down these false positives, we incorporate noise reduction rules based on training data. We leverage two notions: (1) benign prerequisite matches and (2) benign data flow quantity.

Noise reduction based on benign prerequisites. For each process, our system learns prerequisites that fired frequently when the system is run in a benign context. At runtime, when the prerequisites of a triggered TTP match the prerequisites that were encountered during training, we ignore the match.

Noise reduction based on data flow quantity. Filtering based on benign prerequisites may lead to *false negatives:* a malicious event may go unnoticed because it matches behavior observed during the learning phase. For instance, even without any attack, `nginx` reads `/etc/passwd` during its startup phase. However, if we were to whitelist all `nginx` access to `/etc/passwd`, then a subsequent read by a compromised `nginx` server will go unnoticed.

To tackle this problem, we enhance our learning to incorporate quantities of information flow, measured in bytes transferred. For instance, the amount of information that can flow from the file `/etc/passwd` to `nginx` is equal to the size of that file, since `nginx` reads that file only once. Therefore, if significantly more bytes are observed flowing from `/etc/passwd` to `nginx`, then this flow *may* be part of an attack. To determine the cut-off points for information quantity, we observe process-file and process-socket pairs over a period in a benign setting.

4.4.6 Signal Correlation and Detection

Given a set of HSGs, how do we distinguish the ones that constitute an attack with a high confidence? We address this challenge by assigning a severity score to each HSG. This assignment proceeds in two steps further described below.

Threat Tuples. First, we represent the attacker's progress in a campaign by an abstract *threat tuple* associated with the corresponding HSG. In particular, for every HSG, a *threat tuple* is a 7-tuple $\langle S_1, S_2, S_3, ..., S_7 \rangle$ where each S_i corresponds to the severity level of the APT stage at index i of the HSG. We chose 7-tuples based on an extensive survey of APTs in the wild [5], but other choices are possible as well.

Since different TTPs belonging to a certain APT stage may have different severity levels, there are usually multiple candidates to pick from. It is natural to choose the highest severity level among these candidates. For instance, the *threat tuple* associated with the HSG of Figure 8 is $\langle M, L, H, H, -, H, M \rangle$. This tuple contains 6 entries because its matched TTPs belong to 6 different APT stages. The entries are ordered according to the order of the APT stages in the kill-chain. For instance, the first entry of the tuple is M since the most severe TTP belonging to *Initial_Reconnaissance* in the graph has severity M.

HSG Ranking and Prioritization. To rank HSGs, we first transform a threat tuple to a numeric value. In particular, we first map each element of a threat tuple to a numerical value based on the conversion table (Table XIII) included in the Common Vulnerability Scoring System (CVSS), a vendor-neutral industry standard created through the collaboration of security professionals across commercial, non-commercial, and academic sectors [44]. Alternative scoring choices may be made by an enterprise, taking into context its perceived threats and past threat history.

TABLE XIII

NIST severity rating scale

Qualitative level	**Quantitative Range**	**Rounded up Average Value**
Low	0.1 - 3.9	2.0
Medium	4.0 - 6.9	6.0
High	7.0 - 8.9	8.0
Critical	9.0 - 10.0	10.0

Next, we combine the numeric scores for the 7 APT stages into a single overall score. The formula that we use to compute this score was designed with two main criteria in mind: (1) flexibility and customization, and (2) the correlation of APT steps is reflected in the magnification of the score as the steps unfold. To address these criteria, we associate a weight with each entry in the converted threat tuple and calculate a *weighted product* of the threat tuple as the score. These weights are configurable by a system administrator, and they can be used to prioritize detection of specific stages over other stages.

Using a training set, we performed several experiments and compared results using other schemes, such as weighted sum, exponential sum, and geometric sum. For each equation, we measured the average margin between the benign subgraph scores and the attack subgraph

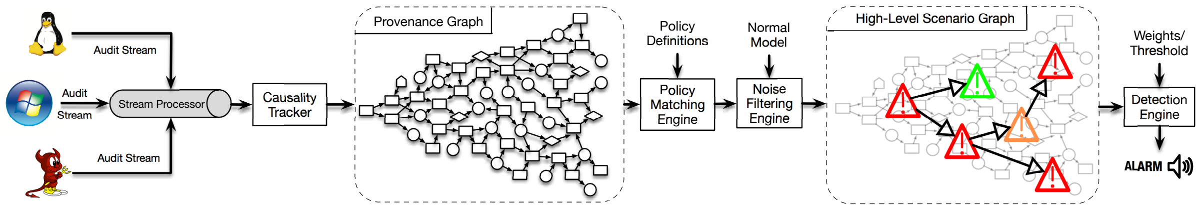


Figure 9. HOLMES Architecture.

scores after normalization and found that the weighted product had the best results. Hence we use the following criteria to flag an APT attack:

$$\prod_{i=1}^{n}(S_i)^{w_i} \geq \tau \tag{4.1}$$

Here, n is the number of APT stages, w_i and S_i denote respectively the weight and severity of stage i, and τ is the detection threshold. If no TTP occurs in stage i, we set $S_i = 1$.

4.5 Implementation

Stream Consumption for Provenance Graph Construction. Figure 9 shows the architecture of HOLMES. To achieve platform independence, audit records from different OSs are normalized to a common data representation (CDR) with shared abstractions for various system entities. For streamlined audit data processing, CDR-based audit records are published to a stream processing server (*Kafka*) and real-time analysis and detection proceeds by consuming from the streaming server. We use our SLEUTH system [1] for stream consumption, causality tracking, and provenance graph construction, so we don't describe those steps in detail here.

Policy Matching Engine and HSG Construction. The *Policy Matching Engine* takes the TTP rule specifications as input and operates on the provenance graph. A representative set of the TTP rule specifications used in the current implementation of HOLMES is shown in Table XIV. To match a TTP, as the provenance graph is being built, the policy matching engine iterates over each rule in the rules table and its prerequisites. A particularly challenging part of this task is to check, for each TTP, the prerequisite conditions about previously matched TTPs and the *path_factor*. In fact, previously matched TTPs may be located in a distant region of the graph and the *path_factor* value may depend on long paths, which must be traversed. We note that a common practice in prior work [1, 9, 25, 40] on attack forensics is to do backward tracking from a TTP matching point to reach an initial compromise point. Unfortunately, this is a computationally expensive strategy in a real-time setting as the provenance graph might contain millions of events.

To solve this challenge without backtracking, we use an incremental matching approach that stores the results of the previous computations and matches and propagates pointers to those results along the graph. When a specific TTP, which may appear as a prerequisite condition in other TTPs, is matched, we create the corresponding node in the HSG and a pointer to that node. The pointer is next propagated to all the low-level entities that have dependencies on the entities of that matched TTP.

The *path_factor* is similarly computed. In particular, given a matched TTP represented as a node in the HSG, a *path_factor* value is incrementally computed for the nodes of the provenance graph that have dependencies on the entities of the matched TTP. Assuming N_1 as a process generating an event matching a TTP, $path\_factor(N_1, N_1)$ is initially assigned to 1. Subsequently, when an edge (N_1, N_2) is added to the provenance graph, $path\_factor(N_1, N_2)$ will be 1 if N_2 is a non-process node or if it is a process with at least one common ancestor with N_1. Otherwise, the $path\_factor$ value increases by 1. In cases that an information flow happens from N_2 to N_3 while both N_2 and N_3 already have a dependency flow from N_1, a new version of N_3 is constructed, and the $path\_factor(N_1, N_{3\_new})$ is set to the minimum among the $path\_factors$ calculated by both flows. Note that in the acyclic provenance graph which is built based on this versioning system, the $path\_factor(N_1, N_2)$ never changes once it is set. Finally, when an event corresponding to a TTP event is encountered, we can reuse the pointer to the prerequisite TTPs and the precomputed *path_factor* immediately if they are available.

An expected bottleneck for this pointer-based correlation of the two layers (provenance graph and HSG) is the space overhead and complexity it adds as the provenance graph grows over time. Our operational observation is that, typically, a large number of entities point to the same set of TTPs; This phenomenon is not random and is actually the result of the propagation of pointers in the process tree, from parent processes to all their descendants. It is, in fact, rare that new pointers get added as the analysis proceeds. In general, the key implementation insight is to maintain an intermediate object that maps entities of the provenance graph to TTPs of the HSG. Therefore, each entity in the provenance graph has only one pointer pointing to the intermediate mapper, and the mapper object contains the set of pointers.

Noise Filtering and Detection Engines. The *Noise Filtering Engine* identifies benign TTP matches so that they can be excluded from the HSG. It takes as input the normal behavior

TABLE XIV: Representative TTPs. Event family denotes a set of corresponding events in Windows, Linux, and FreeBSD. In the Severity column, L=Low, M=Moderate, H=High, C=Critical. Entity types are shown by the characters: P=Process, F=File, S=Socket, M=Memory, U=User.

APT Stage	**TTP**	**Event Family**	**Severity**	**Prerequisites**
$Initial\_Compromise(P)$	$Untrusted\_Read(S, P)$	READ	L	$S.ip \notin$ {Trusted_IP_Addresses}
	$Make\_Mem\_Exec(P, M)$	MPROTECT	M	\$PROT_EXEC\$ $\in M.flags$ $\land \exists\ Untrusted\_Read(?, P')$: $path\_factor(P', P) <= path\_thres$
	$Make\_File\_Exec(P, F)$	CHMOD	H	\$PROT_EXEC\$ $\in F.mode$ $\land \exists\ Untrusted\_Read(?, P')$: $path\_factor(P', F) <= path\_thres$ $\land \exists\ Untrusted\_Read(?, P'')$: $path\_factor(P'', P) <= path\_thres$
	$Untrusted\_File\_Exec(F, P)$	EXEC	C	$\exists\ Untrusted\_Read(?, P') : path\_factor(P', F) <= path\_thres$
$Establish\_Foothold(P)$	$Shell\_Exec(F, P)$	EXEC	M	$F.path \in$ {Command_Line_Utilities} $\land \exists\ Initial\_Compromise(P')$: $path\_factor(P', P) <= path\_thres$
	$CnC(P, S)$	SEND	L	$S.ip \notin$ {Trusted_IP_Addresses} $\land \exists\ Initial\_Compromise(P')$: $path\_factor(P', P) <= path\_thres$
$Privilege\_Escalation(P)$	$Sudo\_Exec(F, P)$	EXEC	H	$F.path \in$ {SuperUser_Tools} $\land \exists\ Initial\_Compromise(P')$: $path\_factor(P', P) <= path\_thres$
	$Switch\_SU(U, P)$	SETUID	H	$U.id \in$ {SuperUser_Group} $\land \exists\ Initial\_Compromise(P')$: $path\_factor(P', P) <= path\_thres$
$Internal\_Recon(P)$	$Sensitive\_Read(F, P)$	READ	M	$F.path \in$ {Sensitive_Files} $\land \exists\ Initial\_Compromise(P')$: $path\_factor(P', P) <= path\_thres$
	$Sensitive\_Command(P, P')$	FORK	H	$P'.name \in$ {Sensitive_Commands} $\land \exists\ Initial\_Compromise(P'')$: $path\_factor(P'', P) <= path\_thres$

Continued on next page

Table XIV – *Continued from previous page*

APT Stage	**TTP**	**Event Family**	**Severity**	**Prerequisites**
$Move\_Laterally(P)$	$Send\_Internal(P, S)$	SEND	M	$S.ip \in$ {Internal_IP_Range} $\wedge \exists\ Initial\_Compromise(P')$: $path\_factor(P', P) <= path\_thres$
$Complete\_Mission(P)$	$Sensitive\_Leak(P, S)$	SEND	H	$S.ip \notin$ {Trusted_IP_Addresses} $\wedge \exists\ Internal\_Reconnaissance(P')$: $path\_factor(P', P) <= path\_thres$ $\wedge \exists\ Initial\_Compromise(P'')$: $path\_factor(P'', P) <= path\_thres$
	$Destroy\_System(F, P)$	WRITE/ UNLINK	C	$F.path \in$ {System_Critical_Files} $\wedge \exists\ Initial\_Compromise(P')$: $path\_factor(P', P) <= path\_thres$
$Cleanup\_Tracks(P)$	$Clear\_Logs(P, F)$	UNLINK	H	$F.path \in$ {Log_Files}$\wedge \exists\ Initial\_Compromise(P')$: $path\_factor(P', P) <= path\_thres$
	$Sensitive\_Temp\_RM(P, F)$	UNLINK	M	$\exists\ Internal\_Reconnaissance(P')$: $path\_factor(P', F) <= path\_thres$ $\wedge \exists\ Initial\_Compromise(P'')$: $path\_factor(P'', P) <= path\_thres$
	$Untrusted\_File\_RM(P, F)$	UNLINK	M	$\exists\ Initial\_Compromise(P')$: $path\_factor(P', F) <= path\_thres$ $\wedge \exists\ Initial\_Compromise(P'')$: $path\_factor(P'', P) <= path\_thres$

model learned on benign runs. This model contains a map of the TTPs that are matched in benign runs and the threshold on the number of bytes read from or written to system objects on these runs. When the policy matching engine matches a new TTP, the entities and prerequisites of that TTP are searched in this model. If an entry exists in the model that contains all the prerequisites and the matched event (having the same entity names), then the total amount of transferred bytes is checked against the benign threshold. If the total amount of bytes transferred is lower than the benign threshold, then the node corresponding to the matched TTP is filtered out; otherwise, a node corresponding to it gets added to the HSG. Finally, the *detection engine* computes the weighted sums of the different HSGs and raises alarms when that value surpasses the detection threshold.

4.6 Experimental Evaluation

Our experimental evaluation is done on red-team vs. blue-team adversarial engagements organized by TC program. We first evaluated HOLMES on a dataset that was available to us beforehand (Sections 4.6.1,4.6.2,4.6.3,4.6.4). Using this evaluation, we calculate the optimal threshold value for HOLMES in Section 4.6.5, and measure its performance in Section 4.6.6. Finally, in Section 4.6.7, we explored applicability of HOLMES as a real-world live detection system in a setting that we have no prior knowledge of when or how red-team is conducting the attacks. After our live experiment, this dataset has been released in the public domain [28] to stimulate further research in this area.

4.6.1 Datasets

Attacks. The datasets we used for evaluation are summarized in Table XV. This table shows nine APT scenarios from 7 hosts across three OS platforms. There are three scenarios for each platform. Collectively, the streams cover 20 days' worth of audit logs. Streams 5 and 7 each contain two independent APT attacks, while the remaining streams contain one APT attack each.

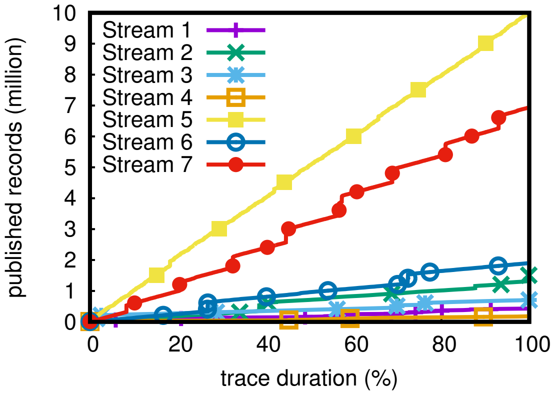
In a nutshell, the adversarial goals and activities in the red team attack scenarios cover those of high-profile APT campaigns. These include typical APT activities such as browser-induced

TABLE XV

Datasets. Streams 5 and 7 contain two independent attack vectors occurring on the same host.

Stream No.	Duration	Platform	Scenario No.	Scenario Name	Attack Surface
1	0d1h17m	Ubuntu 14.04 (64bit)	1	Drive-by Download	Firefox 42.0
2	2d5h8m	Ubuntu 12.04 (64bit)	2	Trojan	Firefox 20.0
3	1d7h25m	Ubuntu 12.04 (64bit)	3	Trojan	Firefox 20.0
4	0d1h39m	Windows 7 Pro (64bit)	4	Spyware	Firefox 44.0
5	5d5h17m	Windows 7 Pro (64bit)	5.1	Eternal Blue	Vulnerable SMB
			5.2	RAT	Firefox 44.0
6	2d5h17m	FreeBSD 11.0 (64bit)	6	Web-Shell	Backdoored Nginx
7	8d7h15m	FreeBSD 11.0 (64bit)	7.1	RAT	Backdoored Nginx
			7.2	Password Hijacking	Backdoored Nginx

drive-by initial compromises, backdoor injection, privilege escalation, internal reconnaissance, exfiltration of sensitive assets, and cleanup of attack footprints. In these attacks, sophisticated attack vectors such as reflective loading, web-shell capabilities, and in-memory module loading were used by a red-team to evaluate the robustness of HOLMES against APT attacks.



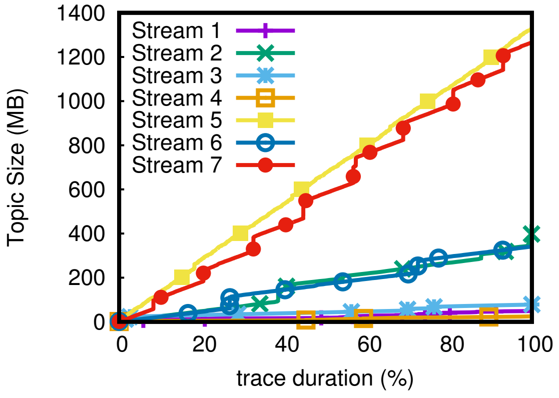


Figure 10. (Left): Number (millions) of published records vs. % of trace duration. (Right): Topic Size (MB) vs. % of trace duration.

Benign data. To build noise reduction models, we used 4 days' worth of benign audit data streamed from the target hosts. These programs include browsers (e.g., Firefox), web servers (e.g., *Nginx*), and various daemons (e.g., *postfix*, *syslogd*).

4.6.2 Setup

Holmes subscribed to the 7 Kafka topics (one topic per host) to consume audit streams and perform analysis and detection in real-time. The number of records published (in millions) with regards to the percentage growth of the audit trace duration is shown in Figure 10 (left). Note that Holmes consumes as fast as the publishing rate from the Kafka server. Figure 10 (right) shows the incremental growth in the size of records published into each Kafka topic.

We configured Holmes with TTPs mentioned in Table XIV and set *path_thres* $= 3$ for prerequisites on TTPs and *weight* $= (10 + i)/10$ for APT stage i, which takes into account slightly higher weights for later APT stages.

4.6.3 Results in a Nutshell

Table XVI summarizes the detection of the nine attack scenarios. The second column shows the *threat tuple* of each HSG matched during detection, and the third column shows the corresponding *threat score*. The fourth column shows the highest score among all benign scenarios of the machine on which the attack scenario is exercised. These benign scenarios might contain the exact programs in the corresponding attack scenario.

The highest score assigned to benign HSGs is 338 (Scenario-3), and the lowest score assigned to attack HSGs is 608 (Scenario-5.2) which is related to an incomplete attack with no harm done to the system. This shows that HOLMES has separated attack and benign scenarios into two disjoint clusters, and makes a clear distinction between them.

TABLE XVI

Scores Assigned to Attack Scenarios. L = Low, M = Moderate, H = High, C = Critical. **Note:** for each scenario, Highest Benign Score in Dataset is the highest *threat score* assigned to benign background activities streamed during the audit log collection of a host (pre-attack, in parallel to attack, and post-attack).

Scenario No.	Threat Tuple	Threat Score	Highest Benign Score in Dataset
1	$\langle C, M, -, H, -, H, M\rangle$	1163881	61
2	$\langle C, M, -, H, -, H, -\rangle$	55342	226
3	$\langle C, M, -, H, -, H, M\rangle$	1163881	338
4	$\langle C, M, -, H, -, -, M\rangle$	41780	5
5.1	$\langle C, L, -, M, -, H, H\rangle$	339504	104
5.2	$\langle C, L, -, -, -, -, M\rangle$	608	
6	$\langle L, L, H, M, -, H, -\rangle$	25162	137
7.1	$\langle C, L, H, H, -, H, M\rangle$	4649220	133
7.2	$\langle M, L, H, H, -, H, M\rangle$	2650614	

The effect of learning noise reduction rules and *path_factor* are shown in Figure 11. This plot shows *threat score* for all benign and attack HSGs which are constructed after analyzing all the seven streams. These scores are shown under three different settings: default which both learning and *path_factor* calculations are enabled, without learning, and without *path_factor*. It is obvious in the figure that with learning and *path_factor*, there is a more considerable margin between attack HSGs and benign ones. Without learning or *path_factor*, we notice an increase in noise, which leads to false positives or false negatives. The 10th percentile, first quartile, and median of default box are all colliding on the bottom line of this box (score= 2.1). This means that more than 50% of *threat scores* are 2.1, which is the result of having many HSGs with only one low severity *Untrusted Read* TTP.

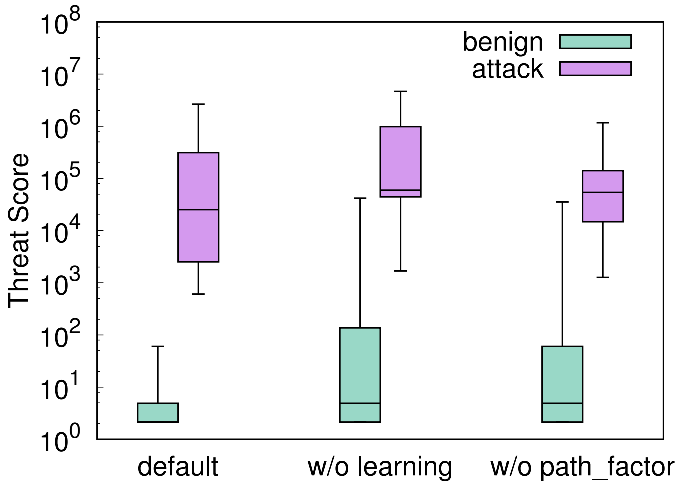


Figure 11. Effects of Learning and *path_factor* on Noise Reduction. Box covers from first to third quartiles while a bar in the middle indicates median, and whisker is extended from 10th to 90th percentile.

4.6.4 Attack Scenarios

We now describe an additional attack scenario detected by Holmes. For reasons of space, we include details of the rest of the scenarios and the related figures in the Appendix B. We note that Scenario-7.2 is discussed in section 4.2 and a portion of its provenance graph and HSG are shown in Figure 6 and Figure 8, respectively.

Scenario-1: Drive-by Download. In this attack scenario (see Figure 12), the user visits a malicious website with a vulnerable Firefox browser. As a result, a file named *net* is dropped and executed on the victim's host. This file, after execution, connects to a C&C server, and a reverse shell is provided to the attacker. The attacker then launches a shell prompt and executes commands such as *hostname*, *whoami*, *ifconfig*, *netstat*, and *uname*. Finally, the malicious executable exfiltrates information to the IP address of the C&C server and then the attacker removes the dropped malicious file.

As can be seen from Figure 12, in the Initial Compromise APT stage, an untrusted file is executed, which matches a TTP with the critical severity level. The final *threat tuple* for this graph looks like $\langle C, M, -, H, -, H, M\rangle$ for all APT stages (see Table XVI). Consequently, the converted quantitative values are $\langle 10, 6, 1, 8, 1, 8, 6\rangle$, which results in a *threat score* equal to 1163881.

4.6.5 Finding the Optimal Threshold Value

To determine the optimal threshold value, we measured the precision and recall by varying threshold values as shown in Figure 13. F-score, the harmonic mean of precision and recall, is maximum at the interval [338.25, 608.26], which is the range from the maximum score of benign subgraphs to the minimum score of attack subgraphs. Therefore, by choosing any threshold in this range, HOLMES makes a clear distinction between attack and benign subgraphs in the tested datasets, with accuracy and recall equal to 1.

To find the optimal value, we first transform the *threat scores* to a linear scale by getting their nth root, where n equals to $\sum_{i=1}^{7} w_i$. The transformed value shows the average contribution

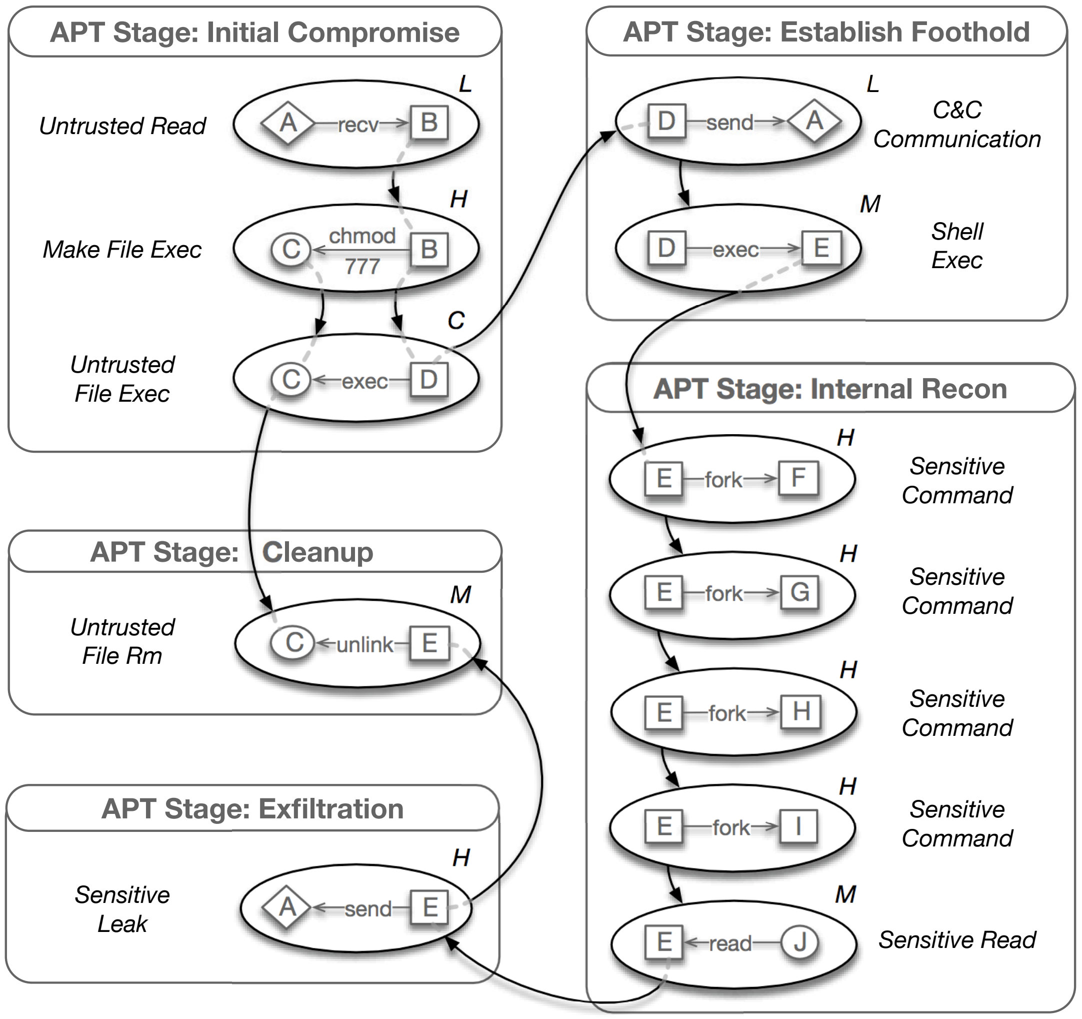


Figure 12. HSG of Scenario-1 (Drive-by Download). Notations: A= Untrusted External Address; B= Firefox; C= Malicious dropped file (net); D= RAT process; E= bash; F= whoami; G= uname; I= netstat; J= company_secret.txt;

of each APT step to the overall *threat score*, and it is a value in the range [1,10]. As all our tested datasets so far belong to single hosts, we exclude the weight of lateral movement step

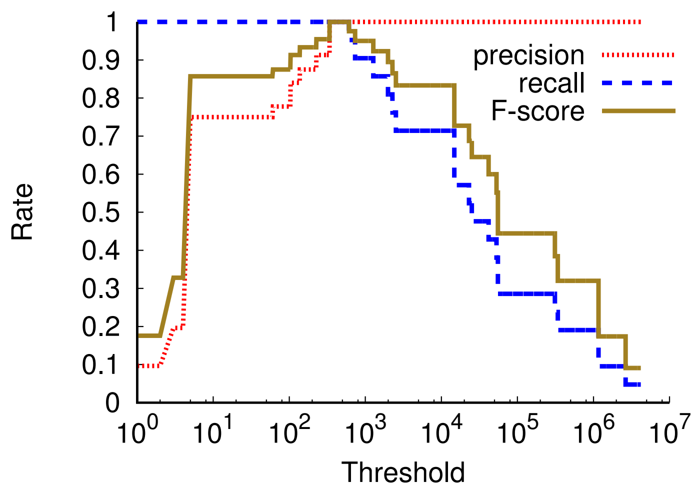


Figure 13. Precision, Recall, and F-score of attack detection by varying the threshold value.

(w_5), which leads to $n = 8.3$. After getting the nth root, the interval of maximum F-score would change to [2.01, 2.16]. Finally, we consider the middle of this range (2.09) as the average severity that each APT step is allowed to contribute to the overall *threat score*, in a benign setting.

4.6.6 Performance

Graph Size. Figure 14 shows the comparison of the growth trends for provenance graph in thousands of edges (left) and the HSG in the number of edges (right). The graph size ratio measured in edges is 1875:1, i.e., an 1875-fold reduction is achieved in the process of mapping from the provenance graph to the HSG.

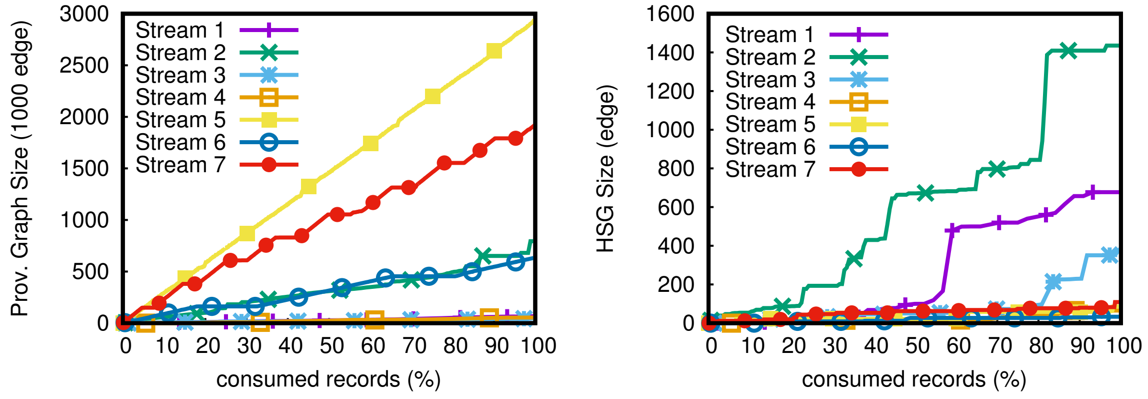


Figure 14. (Left): Provenance graph growth vs. consumed records. (Right): HSG growth vs. consumed records.

Memory Use. HOLMES was tested on an 8 core CPU with a 2.5GHz speed each and a 150GB of RAM. Figure 15 (left) shows the memory consumption of HOLMES with the number of audit records. It shows a nearly linear growth in memory consumption since our system operates on audit records in-memory. Figure 15 (right) shows extrapolation of how many hosts HOLMES can support (regarding memory consumption) with scalability to an enterprise of hundreds of hosts. It is evident that as the number of hosts is increased, the duration that we can keep the full provenance graph in memory decreases. Notice that both x and y-axes are in log-2 scale.

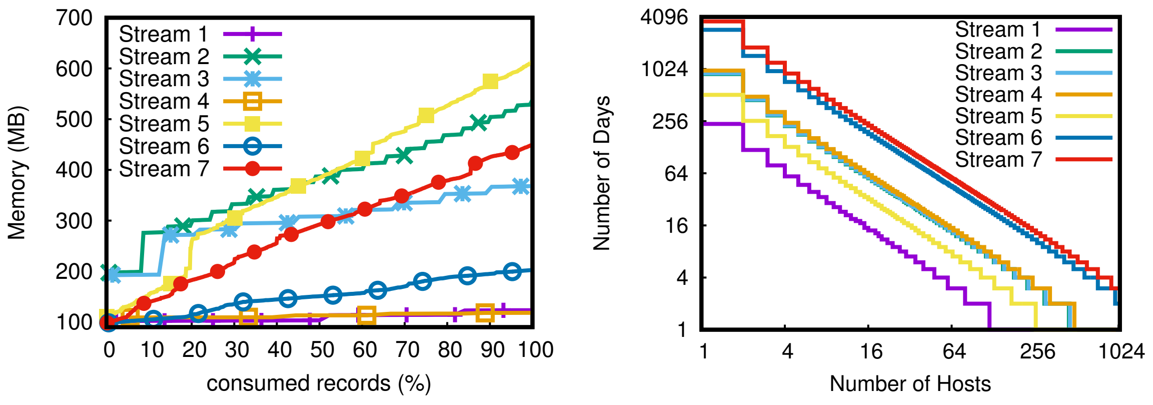


Figure 15. (Left): Memory footprint (MB) vs. % of records consumed. (Right): Number of Days vs. extrapolated number of hosts that can be handled by HOLMES in respect to Memory consumption

Runtime. While HOLMES consumes and analyzes audit records from a Kafka server as the records become available in real-time, to stress-test its performance, we assumed that all the audit records were available at once. Then, we measured the CPU time for consuming the records, building the provenance graph, constructing the HSG, and detecting APTs. We define "CPU Utilization" as the ratio of required CPU time to the total duration of a scenario. In Figure 16, the bars show CPU Utilization for each scenario, and the line shows an extrapolation of how many hosts (of comparable audit trace durations with the scenarios) HOLMES can support if CPU was the limiting factor. This chart shows that our single CPU can support an enterprise with hundreds of hosts.

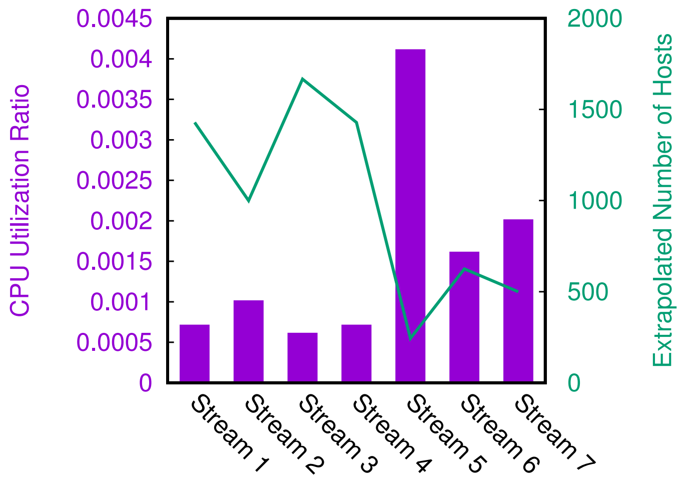


Figure 16. CPU Utilization and the extrapolated number of hosts that can be handled by HOLMES in respect to CPU time.

4.6.7 Live Experiment

To explore how HOLMES would respond to attacks embedded within a predominantly benign stream of events, we evaluated it as a live detection system. This experiment spanned 2 weeks, and during this period, audit logs of multiple systems, running Windows, Linux, or BSD, were collected and analyzed in real-time by HOLMES. In this experiment, an enterprise is simulated with security-critical services such as a web server, E-mail server, SSH server, and an SMB server for providing shared access to files. Similar to the previous datasets, an extensive set of normal activities are conducted during this experiment, and red-team carried out a series of attacks. However, this time, we configured all the parameters beforehand and had no prior knowledge of the attacks planned by the red-team. Moreover, we had cross host internal connectivity, which makes APT stage 5 (Move_laterally) a possible move for attackers. To this end, we set the detection threshold equal to $2.09^{\sum_{i=1}^{7} w_i} = 2.09^{9.8} = 1378$. Figure 17 shows the cumulative distribution function for attack and benign HSGs that HOLMES constructs during this experiment. Note that there are some points representing *threat score* of benign HSGs, that have bypassed the threshold. We explain them as false positives in the following and then discuss some potential false negative scenarios.

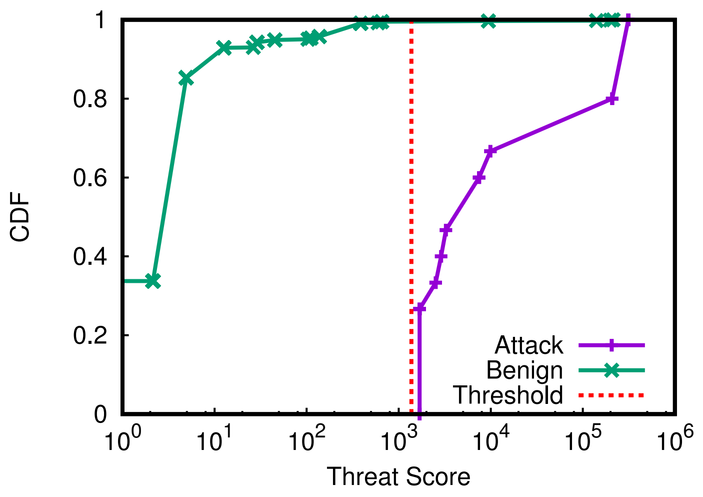


Figure 17. Cumulative distribution function for attack vs. benign HSGs

False Positives. We noticed some false alarms because of SSH connections made by system administrators. These connections come from untrusted IP addresses, and subsequently, Holmes aggregates the severity scores of all the actions issued by the system administrator via an SSH connection. In some cases, the *threat score* bypasses our threshold. The solution is to define a custom tagging policy for servers such as ssh that perform authentication so that the children of such servers aren't marked as untrusted [1].

To further evaluate our system for false alarms, we also evaluated it on another two weeks benign activity period. During this time, a diverse set of normal activities were conducted, (including software updates and upgrades through package managers) and Holmes generated no false alarms.

Based on our results, we claim that the false positive of Holmes is at an acceptable rate considering the benefits it adds to an enterprise. Security analysts can manually check the raised alarms and neutralize HSGs that are falsely constructed.

False Negatives. Although we did not observe any false negatives during our experiments, here we discuss potential scenarios Holmes might miss.

Implicit causality between TTPs: For information flow that avoids system calls, Holmes have no direct visibility to the causal relations between system entities. However, if the rest of the attack unfolds with visibility through system calls, Holmes will still partially reconstruct the attack.

Multiple entry points: As an active evasion technique, attackers might exploit multiple entry points that result in detached subgraphs. Holmes follows every single entry point until our detection threshold is satisfied and correlates TTPs from disjoint subgraphs when there is information flow between them. Nevertheless, some additional analyses might be needed to completely correlate attack steps, which are coming from different entry points and have no information flow in between.

4.7 Summary

We present Holmes, a real-time APT detection system that correlates tactics, techniques, and procedures that might be used to carry out each APT stage. Holmes generates a high-level graph that summarizes the attacker's steps in real-time. We evaluate Holmes against nine real-world APT threats and deploy it as a real-time intrusion detection tool. The results show that Holmes successfully detects APT campaigns with high precision and low false alarm rates.

CHAPTER 5

Poirot: ALIGNING ATTACK BEHAVIOR WITH KERNEL AUDIT RECORDS FOR CYBER THREAT HUNTING

This chapter includes excerpts and figures from material that is published in [3].

5.1 Introduction

In general, threat hunting inside an enterprise presents several challenges:

- *Search at scale*: To remain under the radar, an attacker often performs the attack steps over long periods (weeks, or in some cases, months). Hence, it is necessary to design an approach that can link related IOCs together even if they are conducted over a long period of time. To this end, the system should be capable of searching among millions of log events (99.9% of which often correspond to benign activities).
- *Robust identification and linking of threat-relevant entities*: Threat hunting must be sound in identifying whether an attack campaign has affected a system, even though the attacker might have mutated the artifacts like file hashes and IP addresses to evade detection. Therefore, a robust approach should not merely look for matching IOCs in isolation, but uncover the entire threat scenario, which is harder for an attacker to mutate.
- *Efficient Matching*: For a cyber analyst to understand and react to a threat incident in a timely fashion, the approach must efficiently conduct the search and not produce many

false positives so that appropriate cyber-response operations can be initiated in a timely fashion.

Commonly, knowledge about the malware employed in APT campaigns is published in cyber threat intelligence (CTI) reports and is presented in a variety of forms such as natural language, structured, and semi-structured form. To facilitate the smooth exchange of CTI in the form of IOCs and enable characterization of adversarial techniques, tactics, and procedures (TTPs), the security community has adopted open standards such as OpenIOC [45], STIX [46], and MISP [47]. To provide a better overview of attacks, these standards often incorporate descriptive relationships showing how indicators or observables are related to each other [48].

However, a vast majority of the current threat hunting approaches operates only over fragmented views of cyber threats [11, 12], such as signatures (e.g., hashes of artifacts), suspicious file/process names, and IP addresses (domain names), or by using heuristics such as timestamps to correlate suspicious events [49]. These approaches are useful but have limitations, such as (i) lacking the precision to reveal the complete picture as to how the threat unfolded especially over long periods (weeks, or in some cases, months), (ii) being susceptible to false signals when adversaries use legitimate-looking names (like svchost in Windows) to make their attacks indistinguishable from benign system activities, and (iii) relying on low-level signatures, which makes them ineffective when attackers update or re-purpose [50, 51] their tools or change their signatures (IP addresses or hash values) to evade detection. To overcome these limitations and build a robust detection system, the correlation among IOCs must be taken into account. In fact, the relationships between IOC artifacts contain essential clues on the *behavior* of the attacks inside a compromised system, which is tied to attacker goals and is, therefore, more difficult to change [52, 53].

This chapter formalizes the threat hunting problem from CTI reports and IOC descriptions, develops a rigorous approach for deriving the confidence score that indicates the likelihood of success of an attack campaign, and describes a system called POIROT that implements this approach. In a nutshell, given a graph-based representation of IOCs and relationships among them that expresses the overall behavior of an APT, which we call a *query graph*, our approach efficiently finds an embedding of this *query graph* in a much larger *provenance graph*, which contains a representation of kernel audit logs over a long period of time. Kernel audit logs are free of unauthorized tampering as long as system's kernel is not compromised, and reliably contain relationships between system entities (e.g., processes, files, sockets, etc.), in contrast to its alternatives (e.g., firewall, network monitoring, and file access logs) which provide partial information. We assume that to maintain the integrity of kernel audit logs, a real-time kernel audit storage on a separate and secure log server is used as a precaution against log tampering.

More precisely, we formulate threat hunting as a graph pattern matching (GPM) problem searching for causal dependencies or information flows among system entities that are similar to those described in the *query graph*. To be robust against evasive attacks (e.g., mimicry attacks [54, 55]) which aim to influence the matching, we prioritize flows based on the cost they have for an attacker to produce. Given the NP-completeness of the graph matching problem [56], we propose an approximation function and a novel similarity metric to assess an alignment between the *query* and *provenance* graph.

We test Poirot's effectiveness and efficiency using three different datasets, particularly, red-team/blue-team adversarial engagements performed by TC program [28], publicly available real-world incident reports, and attack-free activities generated by ordinary users. In addition, we simulate several attacks from real-world scenarios in a controlled environment and compare Poirot with other tools that are currently used to do threat hunting. We show that Poirot outperforms these tools. We have implemented different kernel log parsers for Linux, FreeBSD, and Windows, and our evaluation results show that Poirot can search inside graphs containing millions of nodes and pinpoint the attacks in a few minutes.

This chapter is organized as follows: An overall architecture of Poirot appears in Section 5.2. In Section 5.3, we provide the formal details of the graph alignment algorithm. Section 5.4 discusses the evaluation, and we conclude in Section 5.5.

5.2 Approach Overview

A high-level view of our approach is shown in Figure 18. We provide a brief overview of the components of Poirot next, with more detailed discussions relegated to Section 5.3.

5.2.1 Provenance Graph Construction

Poirot currently supports consuming kernel audit logs from Microsoft Windows, Linux, and FreeBSD and constructs a provenance graph in memory, similar to the system described in Chapter 3. To support efficient searching on this graph, we leverage additional methods such as fast hashing techniques and reverse indexing for mapping process/file names to unique node IDs.

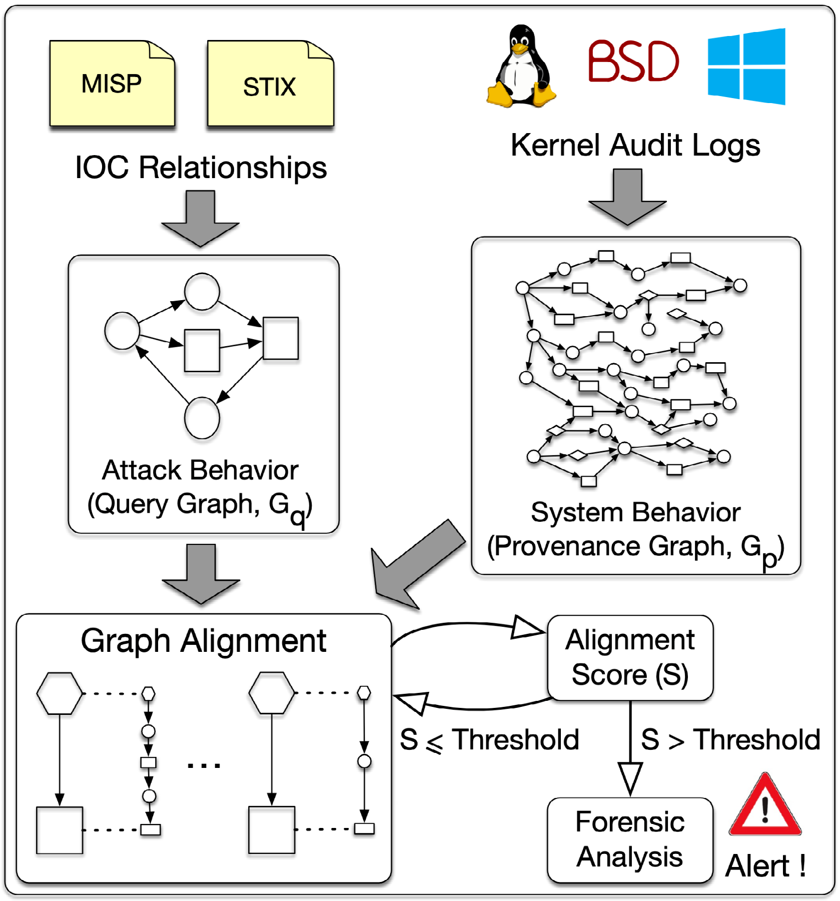


Figure 18. Poirot Approach Overview.

5.2.2 Query Graph Construction

We extract IOCs together with the relationships among them from CTI reports related to a known attack. These reports appear in security blogs, threat intelligence reports by industry, underground forums on cyber threats, and public and private threat intelligence feeds. In addition to natural language, the attacks are often described in structured and semi-structured standard formats as well. These formats include OpenIOC [45], STIX [46], MISP [47], etc. Essentially, these exchange formats are used to describe the salient points of the attacks, the observed IOCs, and the relationships among them. For instance, using OpenIOC the behavior of a malware sample can be described as a list of artifacts such as the files it opens, and the DLLs it loads [57]. These standard descriptions are usually created by the security operators manually [58, 59]. Additionally, automated tools have also been built to automatically extract IOCs from natural language and complement the work of human operators [60–62]. These tools can be used to perform an initial extraction of features to generate the query graph and later refined manually by a security expert. We believe that manual refinement is an important component of the query graph construction because automated methods may often generate noise and reduce the quality of the query graphs.

We model the behavior appearing in CTI reports also as a labeled, typed, and directed graph, which we call *query graph (G_q)*. If a description in a standard format is present, the creation of the query graph can be easily automated and further refined by humans. In particular, the entities appearing in the reports (e.g., processes, files) are transformed into nodes while relationships are transformed into directed edges [63]. Nodes and edges of the query graph may be further associated with additional information such as labels (or names), types (e.g., processes, files, sockets, pipes, etc) and other annotations (e.g., hash values, creation time, etc) depending on the information that an analyst may deem necessary for matching. In the current POIROT implementation, we use names and types for specifying explicit mappings between nodes in the query graph and nodes in the provenance graph.

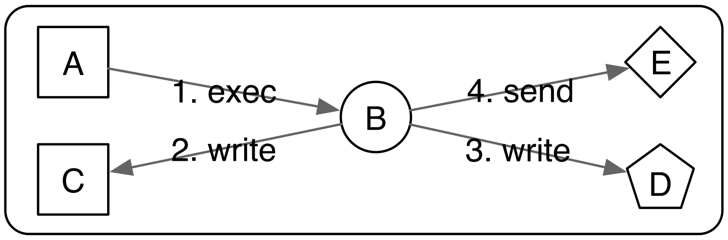


Figure 19. Query Graph of DeputyDog Malware. A=∗.%exe%, B=∗, C=%APPDATA%\∗, D=%HKCU%\Software\Microsoft\Windows\CurrentVersion\ Run\∗, E=%External IP address%.

As an example of query graph construction, consider the following excerpt from a report [64] about the DeputyDog malware, used in our evaluation.

> Upon execution, 8aba4b5184072f2a50cbc5ecfe326701 writes “28542CC0.dll” to this location: “C:\Documents and Settings\All Users\Application Data\28542CC0.dll”. In order to maintain persistence, the original malware adds this registry key: “%HKCU%\Software\Microsoft\Windows\CurrentVersion\Run\ 28542CC0”. The malware then connects to a host in South Korea (180.150.228.102).

The excerpt mentions several actions and entities that perform them and is readily transformed into a graph by a security analyst. For instance, the first sentence clearly denotes a process writing to a file (upon execution the malware writes a file to a location). We point out that the level of detail present in this excerpt is common across a large majority of CTI reports and can be converted to a reliable query graph by a qualified cyber analyst. In particular, the verbs that express actions carried out by subjects can often be easily mapped to reads/writes from/to disk or network and to interactions among processes (e.g., a browser downloads a file, a process spawns another process, a user clicks on a spear-phishing link, etc).

Figure 19 shows the query graph corresponding to the above excerpt. Ovals, diamonds, rectangles, and pentagons represent processes, sockets, files, and registry entries, respectively. In Figure 19, node B represents the malware process or group of processes (we use a $*$ to denote that it can have any name), node A represents the image file of the malware, while nodes C, D and E represent a dropped file, a registry and an Internet location, respectively. We highlight at this point that the query graph that is built contains only information about information flows among specific entities as they appear in the report (processes, files, IP addresses, etc) and is not intended to be a precise subgraph of all the malicious entities that actually appear during the attack. In a certain sense, the query graph is a *summary* of the actual attack graph. In our experiments, the query graphs we obtained were usually small, containing between 10-40 nodes and up to 150 edges.

5.2.3 Graph Alignment

Finally, we model threat hunting as determining whether the query graph G_q for the attack "manifests" itself inside the provenance graph G_p. We call this problem *Graph Alignment Problem.*

We note at this point that G_q expresses several high-level flows between the entities (processes to files, etc.). In contrast, G_p expresses the complete low-level activity of the system. As a result, an edge in G_q might correspond to a path in G_p consisting of multiple edges. For instance, if G_q represents a compromised browser writing to a system file, in G_p this may correspond to a path where a node representing a Firefox process forks new processes, only one of which ultimately writes to the system file. Often, this kind of correspondence may be created by attackers adding noise to their activities to escape detection. Therefore, we need a graph alignment technique that can *match single edges in G_q to paths in G_p*. This requirement is critical in the design of our algorithm.

In graph theory literature, there exist several versions of the graph matching problem. In *exact matching*, the subgraph embedded in a larger graph G_p must be isomorphic to G_q [65]. In contrast, in the *graph pattern matching* (GPM) problem, some of the restrictions of exact matching are relaxed to extract more useful subgraphs. However, both problems are NP-complete in the general case [56]. Even though a substantial body of work dedicated to *GPM* exists [66–72], many have limitations that make them impractical to be deployed in the field of *threat hunting*. Specifically, they (i) are not designed for directed graphs with labels and types assigned to each node, (ii) do not scale to millions of nodes, or (iii) are designed to align all nodes or edges in the query graph exhaustively. Moreover, these approaches are not intended for the context of threat hunting, taking into account an evasive adversary which tries to remain stealthy utilizing the knowledge of the underlying matching criteria. Due to these considerations, we devise a novel graph pattern matching technique that addresses these limitations.

In Figure 18, graph nodes are represented in different shapes to model different node types, such as a file, process, and socket, however, the labels are omitted for brevity. In particular, Poirot starts by finding the set of all possible candidate alignments $i : j$ where i and j represent nodes in $V(G_q)$ and $V(G_p)$, respectively. Then, starting from the alignment with the highest likelihood of finding a match, called a *seed node*, we expand the search to find further node alignments. The seed nodes are represented by hexagons in Figure 18 while matching nodes in the two graphs are connected by dotted lines. To find an alignment that corresponds to the attack represented in CTI relationships, the search is expanded along paths that are more likely to be under the influence of an attacker. To estimate this likelihood, we devise a novel metric named *influence score*. Using this metric allows us to largely exclude irrelevant paths from the search and efficiently mitigate the *dependency explosion* problem. Prior works have also proposed approaches to prioritize flows based on a *score* computed as length [70, 71] or cost [1]. However, they can be defeated by attacks [54,55] in which attackers frequently change their ways to evade the detection techniques. For instance, a proximity-based graph matching approach [70,71] might be easily evaded by attackers, who, being aware of the underlying system and matching approach, might generate a long chain of fork commands to affect the precision of proximity-based graph matching. In contrast, our score definition explicitly takes the influence of a potential attacker into account. In particular, we increase the cost for the attacker to evade our detection, by prioritizing flows based on the effort it takes for an attacker to produce them. Our search for alignment uses such prioritized flows and is described in Section 5.3.

After finding an alignment $G_q :: G_p$, a score is calculated, representing the similarity between G_q and the aligned subgraph of G_p. When the score is higher than a threshold value, POIROT raises an alert which declares the occurrence of an attack and presents a report of aligned nodes to a system analyst for further forensic analysis. Otherwise, POIROT starts an alignment from

TABLE XVII

Notations.

Notation	**Description**
$i : k$	**Node alignment**. Node i is aligned to node k (i and k are in two distinct graphs).
$i \dashrightarrow j$	**Flow**. A path starting at node i and ending at node j.
$i \xrightarrow{\text{label}} j$	An edge from node i to node j with a specific label.
$G_q :: G_p$	**Graph alignment**. A set of node alignments $i : k$ where i is a node of G_q and k is a node of G_p.
$V(G)$	Set of all vertices in graph G.
$E(G)$	Set of all edges in graph G.
$F(G)$	Set of all flows $i \dashrightarrow j$ in graph G such that $i \neq j$.

the next seed node candidate. After finding an attack subgraph in G_p, POIROT generates a report containing the aligned nodes, information flows between them, and the corresponding timestamps. In an enterprise setting, such visually compact and semantic-rich reports provide actionable intelligence to cyber analysts to plan and execute cyber-threat responses. We discuss the details of our approach in Section 5.3.

5.3 Algorithms

In this section, we discuss our main approach for alignment between G_q and G_p by (a) defining an *alignment metric* to measure how proper a graph alignment is, and (b) designing a best-effort similarity search based on specific domain characteristics.

5.3.1 Alignment Metric

We introduce some notations (in Table XVII), where we define two kinds of alignments, i.e., a *node alignment* between two nodes in two different graphs, and a *graph alignment* which is a set of node alignments. Typically, two nodes i and j are in a *node alignment* when they represent the same entity, e.g., a node representing a commonly used browser mentioned in the CTI report (node *%browser%* in the query graph G_q of Figure 20) and a node representing

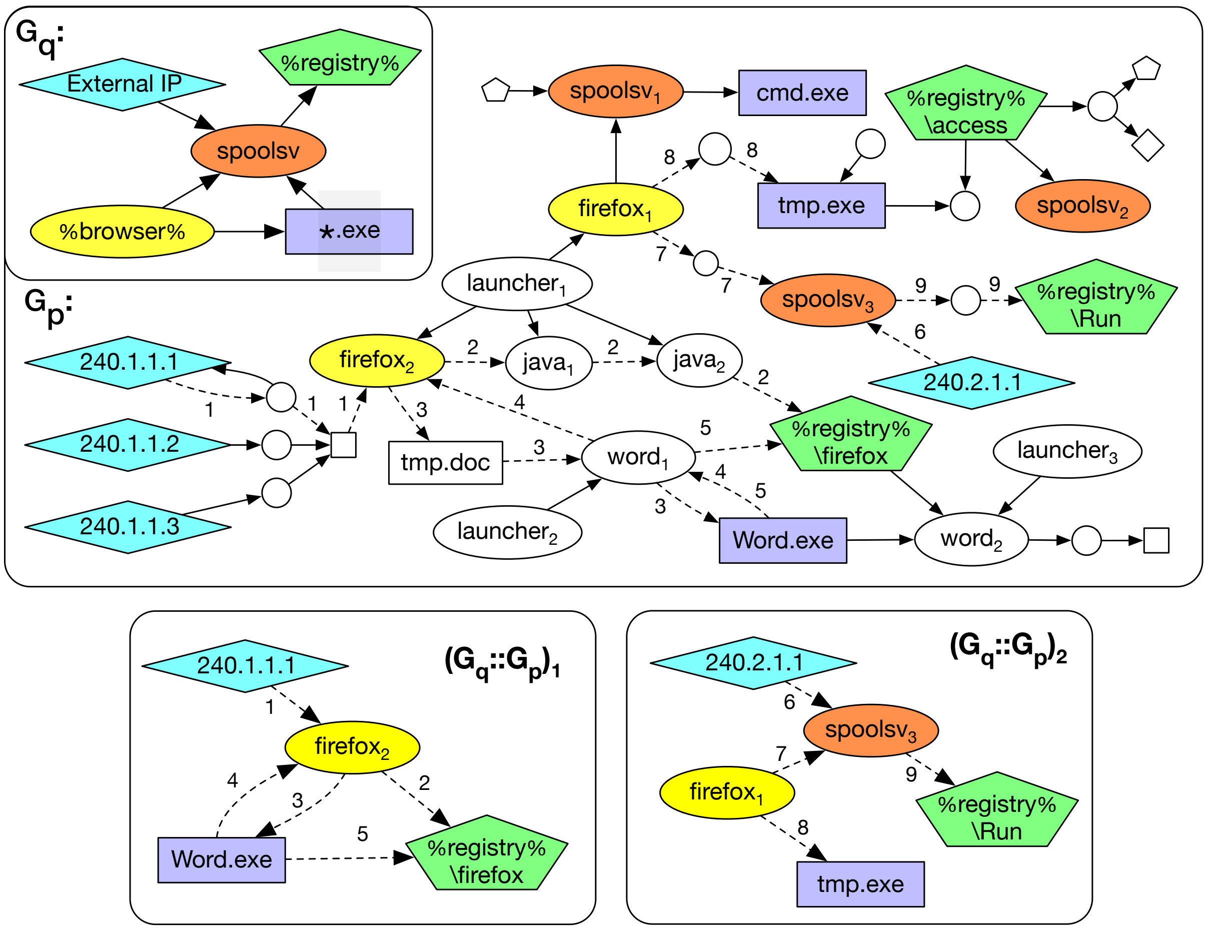


Figure 20. Simplified Provenance Graph (G_p), Query Graph (G_q), and two sample graph alignments ($G_q :: G_p$). Node types are shown with different shapes, and possible alignments for each node is shown with the same color. The numbers on the edges are merely to illustrate possible paths/flows and do not have additional meaning.

a *Firefox* process in the provenance graph. We note that, in general, the node alignment relationship is a many-to-many relationship from V(G_q) to V(G_p), where V(G_q) and V(G_p) are the set of vertices of G_q and G_p respectively. Therefore, given a query graph G_q, there may be a large number of *graph alignments* between G_q and many subgraphs of G_p. Another thing to point out is that each of these *graph alignments* can correspond to different subgraphs of G_p. Each of these subgraphs contains the nodes that are aligned with the nodes of G_q; however, they may contain different paths among those nodes. Among these subgraphs, we are interested in finding the subgraph that best matches the graph G_q.

Based on these definitions, the problem is to find the best possible graph alignment among a set of candidate graph alignments. To illustrate this problem, consider the query and provenance graphs G_q and G_p, and two possible aligned graphs in Figure 20, where the node shapes represent entity types (e.g., process, file, socket), and the edges represent information flow (e.g., read, write, IPC) and causal dependencies (e.g., fork, clone) between nodes. The numbers shown on the edges of G_p are not part of the provenance graph but serve to identify a single path in our discussion. In addition, the subgraphs of G_p determined by these two graph alignments with G_q are represented by dotted edges in G_p. Each flow in G_p and corresponding edge in G_q is labeled with the same number. The problem is, therefore, to decide which among many alignments is the best candidate. Intuitively, for this particular figure, alignment $(G_q :: G_p)_2$ is closer to G_q than $(G_q :: G_p)_1$, mainly because the number of its aligned nodes is higher than that of $(G_q :: G_p)_1$, and most importantly, its flows have a better correspondence to the edges of the query graph G_q.

5.3.1.1 Influence Score

Before formalizing the intuition expressed above, we must introduce a path scoring function, which we call *influence score* and which assigns a number to a given flow between two nodes.

This score will be instrumental in defining the "goodness" of a graph alignment. In practice, the *influence score* represents the likelihood that an attacker can produce a flow. To illustrate this notion, consider the two nodes $\textit{firefox}_2$ and *%registry%\firefox* in the graph G_p in Figure 20. There exist two flows from $\textit{firefox}_2$ to *%registry%\firefox*, one represented by the edges labeled with the number 2 (and passing through nodes java_1 and java_2), and another represented by the edges labeled 3, 3, and 5 (and passing through nodes tmp.doc and word_1). Assuming $\textit{firefox}_2$ is under the control of an attacker, it is more likely for the attacker to execute the first flow rather than the second flow. In fact, in order to exercise the second flow, an attacker would have to take control over process launcher_2 or word_1 in addition to $\textit{firefox}_2$. Since launcher_2 or word_1 share no common ancestors in the process tree with $\textit{firefox}_2$, such takeover would have to involve an additional exploit for launcher_2 or word_1, which is far more unlikely than simply exercising the first flow, where all processes share a common ancestor launcher_1. We point out that this likelihood does not depend on the length of the flow, rather on the number of processes in that flow and on the number of distinct ancestors those processes share in the process tree. One can, in fact, imagine a long chain of forked processes, which are however all under the control of the attacker because they all share a common ancestor in the process tree, i.e., the first process of the chain. Another possible scenario of attacks present in the wild involves remote code loading from a single compromised process, where all the code with malicious functionality is loaded in main memory and the same process (e.g., firefox) executes all the actions on behalf of the attacker. While this technique leaves no traces on the file system and may evade some detection tools, POIROT would be able to detect this kind of attack. In fact the influence score remains trivially unchanged.

One additional important point to note is that this notion of measuring the potential *influence* of an attacker is very robust concerning evasion methods from an attacker. Every activity that an attacker may use to add noise and try to evade detection will likely have the same common ancestors, namely the initial compromise points of the attack, unless the attacker pays a higher cost to perform more distinct compromises. Thus, such efforts will be ineffective in changing the influence score of the paths.

Based on these observations, we define the *influence score*, $\Gamma_{i,j}$, between a node i and a node j as follows:

$$\Gamma_{i,j} = \begin{cases} \max\limits_{i \dashrightarrow j} \dfrac{1}{AC_{min}(i \dashrightarrow j)} & \exists i \dashrightarrow j \mid AC_{min}(i \dashrightarrow j) \leq AC_{thr} \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

In above equation, we use the notion of minimum ancestral cover ($AC_{min}(i \dashrightarrow j)$) which is defined in 4.4.4 and represents the minimum number of distinct, independent compromises an attacker has to conduct to be able to generate the flow $i \dashrightarrow j$. This value captures the extent of the attacker's control over the flow and is calculated based on the minimum number of common ancestors of the processes present in the flow. For instance, if there is a flow from a node i to a node j, and all the processes involved in that flow have one common ancestor in the process tree, an attacker needs to compromise only that common ancestor process to initiate such flow, and therefore $AC_{min}(i \dashrightarrow j)$ is equal to 1. Note that if a node i represents a process and node j a child process of i, then $AC_{min}(i \dashrightarrow j)$ will be equal to 1 as i is parent of j. If the number of common ancestors is larger than one (e.g., there are two ancestors in the path $\mathit{firefox}_2 \rightarrow \mathit{tmp.doc} \rightarrow \mathit{word}_1 \rightarrow \mathit{\%registry\%\backslash firefox}$), an attacker has to compromise at least as many (unrelated) processes independently; therefore it is harder for the attacker to construct such flow. For instance, for the attacker to control the flow $\mathit{firefox}_2 \rightarrow \mathit{tmp.doc} \rightarrow \mathit{word\_1} \rightarrow$ $\mathit{\%registry\%\backslash firefox}$), (s)he needs to control both launcher$_1$ and launcher$_2$; therefore AC_{min} is equal to 2.

We also reasonably assume that it is not practical for an attacker to compromise more than a small number of processes with distinct exploits. In a vast majority of documented APTs, there is usually a single entry point or a very small number of entry points into a system for an attacker, e.g., a spear phishing email, or a drive-by download attack on the browser. We have confirmed that this is true also based on a review of a large number of white papers on APTs [5]. Once an attacker has an initial compromise, it is highly unlikely that they will invest additional resources in discovering and exploiting extra entry points. Therefore, we can place a limit AC_{thr} on the $AC_{min}(i \dashrightarrow j)$ values and reasonably assume that any flow between two nodes that has a $AC_{min}(i \dashrightarrow j)$ greater than AC_{thr} can not have been initiated by an attacker.

While the value of $AC_{min}(i \dashrightarrow j)$ expresses how hard it is for an attacker to control a specific path, the *influence score* expresses how easy it is for an attacker to control that path, and it is defined as a value that is inversely proportional to $AC_{min}(i \dashrightarrow j)$. If there is more than one flow between two nodes i and j with AC_{min} less than or equal to AC_{thr}, the *influence score* will be the maximum $\frac{1}{AC_{min}(i \dashrightarrow j)}$ over all those flows. Therefore, if there exists a path with

$AC_{min}(i \dashrightarrow j)$ less than or equal to AC_{thr} from i to j, the *influence score* would be inverse of *path_factor* which is defined in 4.4.4; otherwise, the *influence score* would be equal to 0. Given that, the value of $\Gamma_{i,j}$ is maximal (equal to 1) when there is a flow whose $AC_{min}(i \dashrightarrow j)$ equals 1 and is minimal (equal to 0) when there is no flow with a $AC_{min}(i \dashrightarrow j)$ less than or equal to AC_{thr}.

5.3.1.2 Alignment Score

We are now ready to define a metric that specifies the score of a graph alignment $G_q :: G_p$. Based on the notion of *influence score*, we define the scoring function $S(G_q :: G_p)$, representing the score for an alignment $G_q :: G_p$ as follows:

$$S(G_q :: G_p) = \frac{1}{|F(G_q)|} \sum_{(i \dashrightarrow j) \in F(G_q)} \Gamma_{k,l} \mid i : k \ \& \ j : l \tag{5.2}$$

In Equation 5.2, nodes i and j are members of $V(G_q)$, and nodes k and l are members of $V(G_p)$. The flow $i \dashrightarrow j$ is a flow defined over G_q. In particular, the formula starts by computing the sum of the influence scores among all the pairs of nodes (k, l) with at least one path from k to l in the graph G_p such that k is aligned with i and l is aligned with j. This sum is next normalized by dividing it with the maximal value possible for that sum. In fact, $|F(G_q)|$ is the number of flows in G_q. Since the maximal value of the *influence score* between two nodes is equal to 1, then the number of flows automatically represents the maximal value of the sum of the *influence scores.*

From Equation 5.2, intuitively, the larger the value of $S(G_q :: G_p)$, the larger the number of node alignments and the larger the similarity between flows in G_q and flows in G_p, which are likely to be under the influence of a potential attacker. In particular, the value of $S(G_q :: G_p)$ is between 0 and 1. When $S(G_q :: G_p) = 0$, either no proper alignment is found for nodes in $V(G_q)$, or no similar flows to those of G_q appear between the aligned nodes in G_p. On the contrary, when $S(G_q :: G_p) = 1$, all the nodes in G_q are aligned to the nodes in G_p, and all the flows existing in G_q also appear between the aligned nodes in G_p, and they all have an *influence score* equal to 1, i.e., it is highly likely that they are under the attacker's control.

Finally, when the alignment score $S(G_q :: G_p)$ bypasses a predetermined threshold value (τ), we raise the alarm. To determine the optimal value of this threshold, recall that AC_{thr} is the maximum number of distinct entry point processes we are assuming an attacker is willing to exploit independently. Therefore, an attacker is assumed to be able to influence any information flow with *influence score* of $\frac{1}{AC_{thr}}$ or higher. On the other hand, $S(G_q :: G_p)$ is the average of all *influence scores*. Therefore, we define the threshold τ as follows:

$$S(G_q :: G_p) \geq \tau \tag{5.3}$$

$$\tau = \frac{1}{AC_{thr}} \tag{5.4}$$

If $S(G_q :: G_p)$ bypasses τ, we declare a match and raise the alarm.

5.3.2 Best-Effort Similarity Search

After defining the alignment score, we describe our procedure to search for an alignment that maximizes that score. In particular, given a query graph G_q, we need to search a very large provenance graph G_p to find an alignment $G_q :: G_p$ with the highest alignment score based on Equation 5.2.

The first challenge in doing this is the size of G_p, which can reach millions of nodes and edges. Therefore, it is not practical to store *influence scores* between all pairs of nodes of G_p. We need to perform graph traversals on demand to find the *influence scores* between nodes or even to find out whether there is a path between two nodes. Besides, we are assuming that all analytics are being done on a stationary snapshot of G_p, and no changes happen to its nodes or edges from the moment when a search is initiated until it terminates.

Our search algorithm consists of the following four steps, where steps 2-4 are repeated until finding alignment with a score higher than the threshold value τ (Equation 5.4).

Step 1. Find all Candidate Node Alignments: We start by searching among nodes of G_p to find candidate alignments for each node in G_q. These candidate alignments are chosen based on the name, type, and annotations on the nodes of the query graph. For instance, nodes of the same type (e.g., two process nodes) with the same label (e.g., Firefox) appearing in G_q and G_p may form candidate alignments, nodes whose labels match a regular expression (e.g., a file system path and file name), and so on. A user may also manually annotate a node in the provenance graph and explicitly specify an alignment with a node in the query graph. In general, a node in G_q may have any number of possible alignments in G_p, including 0. Note that in this first step, we do not have enough information about paths and flows and are looking at nodes in isolation. In Figure 20, the candidate node alignments are represented by the pairs of nodes having the same color.

Step 2. Selecting Seed Nodes: To find a good-enough alignment $G_q :: G_p$, we need to explore connections between candidate alignments found in Step 1, by performing graph traversals on G_p. However, due to the structure and large size of G_p, starting a set of graph traversals from randomly aligned nodes in G_p might lead to costly and unfruitful searches. To determine a *good* starting point, a key observation is that the attack activities usually comprise a tiny portion of G_p, while benign activities are usually repeated multiple times. Therefore, it is more likely for artifacts that are specific to an attack to have fewer alignments than artifacts of benign activities. Based on this observation, we sort the nodes of G_q by an increasing order in the number of candidate alignments related to each node. We select the seed nodes with fewest alignments first. For instance, with respect to the example in Figure 20, the seed node will be *%browser%*, since it has the smallest number of candidate node alignments. If there are seed nodes with the same number of candidate alignments, we choose one of them randomly.

Step 3. Expanding the Search: In this step, starting from the seed node chosen at Step 2, we iterate over all the nodes in G_p aligned to it and initiate a set of graph traversals, going forward or backward, to find out whether we can reach other aligned nodes among those found in Step 1. For instance, after choosing node *%browser%* as a seed node, we start a series of forward and backward graph traversals from the nodes in G_p aligned to *%browser%*, that is $firefox_1$ and $firefox_2$. In theory, these graph traversals can be very costly both because of the size of the graph and also the number of candidate aligned nodes, which can be located anywhere in the graph. In practice, however, we can stop expanding the search along a path once the *influence score* between the seed node and the last node in that path reaches 0. For instance, suppose we decide that AC_{thr} is equal to 2 in Figure 20. Then, the search along the path ($firefox_2 \rightarrow tmp.doc \rightarrow word_1 \rightarrow \%registry\%\backslash firefox \rightarrow word_2$) will not expand past the node word$_2$, since the AC_{min} between *firefox*$_2$ and any node along that path becomes greater than 2 at word$_2$, and thus the influence score becomes 0. Note that there is an additional path from *firefox*$_2$ to word$_2$ via *%registry%\firefox* and along this path, the AC_{min} between *firefox*$_2$ and word$_2$ is still 2. Therefore, because of this path, the search will continue past word$_2$. Using the *influence score* as an upper bound in the graph traversals dramatically reduces the search complexity and enables a fast exploration of the graph G_p.

Based on the shape of the query graph G_q, multiple forward/backward tracking cycles might be required to visit all nodes (for instance, if we choose *%browser%* as a seed node in our example, then node *240.2.1.1* in G_p is unreachable with only one forward or backward traversal starting at *firefox*$_1$ or *firefox*$_2$). In this case, we repeat the backward and forward traversals starting from nodes that are adjacent to the unvisited nodes but that have been visited in a previous traversal (for instance, node *spoolsv*$_3$ in our example). We iterate this process until we cover all the nodes of the query graph G_q.

Step 4. Graph Alignment Selection: This step is responsible for producing the final result or for starting another iteration of the search from step 2, in case a result is not found. In particular, after performing backward/forward traversals, we identify a subset of candidate nodes in G_p for each node in G_q. For instance, with respect to our example, we find that node *%browser%* has candidates *firefox*$_1$ and *firefox*$_2$, node *External IP* has candidate alignments *240.1.1.1*, *240.1.1.2*, *240.1.1.3*, and *240.2.1.1*, and so on. However, the number of possible candidate *graph alignments* that these candidate nodes can form can be quite large. If each node i in G_q has n_i candidate alignments, then the number of possible graph alignments is equal to $\prod_i n_i$. For instance, in our example, we can have 216 possible graph alignments $(2 \times 3 \times 3 \times 3 \times 4)$. In this step, we search for the graph alignment that maximizes the alignment score (Equation 5.2).

A naive method for doing this is a brute-force approach that calculates the alignment score for all possible graph alignments. However, this method is very inefficient and does not fully take advantage of domain knowledge. To perform this search efficiently, we devise a procedure that iteratively chooses the best candidate for each node in G_q based on an approximation function that measures the maximal contribution of each alignment to the final alignment score.

In particular, starting from a seed node in G_q, we select the node in G_p that maximizes the contribution to the alignment score and fix this node alignment (we discuss the selection function in the next paragraph). For instance, starting from seed node *%Browser%* in our example, we fix the alignment with node $\mathit{firefox}_1$. From this fixed node alignment, we follow the edges in G_q to fix the alignment of additional nodes connected to the seed node. The specific node alignment selected for each of these nodes is the one that maximizes the contribution to the alignment score. For instance, from node *%Browser%* (aligned to $\mathit{firefox}_1$), we can proceed to node $*.exe$ and fix the alignment of that with one node among *cmd.exe*, *tmp.exe*, and *Word.exe*, such that the contribution to the alignment score is maximized.

Selection Function. The key intuition behind the selection function, which selects and fixes one among many node alignments, is to approximate how much each alignment would contribute to the final alignment score and to choose the one with the highest contribution. For a given candidate aligned node k in G_p, this contribution is calculated as the sum of the maximum influence scores between that node and all the other candidate nodes l in G_p that: 1) are reachable from k or that have a path to k, and 2) whose corresponding aligned node j in G_q has a flow from/to the node in G_q that corresponds to node k. For instance, consider node *%Browser%* and the two candidate alignment nodes *firefox*$_1$ and *firefox*$_2$ in our example. To determine the contribution of *firefox*$_1$, we measure for every flow (*%Browser%* $\dashrightarrow$ $*$*.exe*, *%Browser%* $\dashrightarrow$ *spoolsv*, *%Browser%* $\dashrightarrow$ *%registry%*) from/to *%Browser%* in G_q, the maximum *influence score* between *firefox*$_1$ and the candidate nodes aligned with $*$*.exe*, *spoolsv*, and *%registry%*, respectively. In other words, we compute the maximum *influence score* between *firefox*$_1$ and each of the node alignment candidates of $*$*.exe*, the maximum *influence score* between *firefox*$_1$ and each of the node alignment candidates of *spoolsv*, and the maximum *influence score* between *firefox*$_1$ and each of the node alignment candidates of *%registry%*. Each of these three maximums provides the maximal contribution to the alignment score of each of the possible future alignments (which are not fixed yet) for $*$*.exe*, *spoolsv*, and *%registry%*, respectively. Next, we sum these three maximum values to obtain the maximal contribution that *firefox*$_1$ would provide to the alignment score. We repeat the same procedure for *firefox*$_2$ and, finally select the alignment with the highest contribution value. This contribution is formally computed by the following equation, which approximates $A(i : k)$ the contribution of a node alignment $i : k$.

$$\begin{aligned}
&A(i : k) \\
&= \sum_{j:(i \dashrightarrow j) \in F(G_q)} \Big(1_{\{j:l\}} \times \Gamma_{k,l} + (1 - 1_{\{j:l\}}) \times \max_{m \in candidates(j)} (\Gamma_{k,m}) \Big) \\
&+ \sum_{j:(j \dashrightarrow i) \in F(G_q)} \Big(1_{\{j:l\}} \times \Gamma_{l,k} + (1 - 1_{\{j:l\}}) \times \max_{m \in candidates(j)} (\Gamma_{m,k}) \Big)
\end{aligned} \tag{5.5}$$

where $1_{\mathcal{A}}$ is an indicator function, which is 1 if the alignment expressed in $\mathcal{A}$ is fixed, and is 0 otherwise. In other words, if the alignment between node j and l, has been fixed, $l_{\{j:l\}}$ equals to 1, and otherwise, if node j is not aligned to any node yet, $1_{\{j:l\}}$ equals to 0. Note that $1_{\{j:l\}}$ and $(1 - 1_{\{j:l\}})$ are mutually exclusive, and at any moment, only one of them equals 1, and the other one equals to 0.

We note that the first summation is performed on outgoing flows from node i, while the second summation is performed on flows that are incoming to node i. Inside each summation, the first term represents a fixed alignment while the second term represents the maximum among potential alignments that have not been fixed yet, as discussed above.

Finally, for each node i having a set K of candidate alignments as produced by Step 3, the selection function, which fixes the alignment of i is as follows:

$$\arg\max_{k \in K} A(i : k) \tag{5.6}$$

The intuition behind equations Equation 5.5 and Equation 5.6 is that once a node alignment is fixed, the other possible alignments of that node are ignored by future steps of the algorithm and the calculation of the maximum influence score related to that alignment is reduced to a table lookup instead of an iteration over candidate node alignments. In particular, the search starts as a brute force search, but as more and more node alignments are fixed, the search becomes faster by reusing results of previous searches stored in the table. Using equations Equation 5.5 and Equation 5.6 dramatically speeds up the determination of a proper graph alignment. While in theory, this represents a greedy approach, which may not always lead to the best results, in practice, we have found that it works very well.

Finally, after fixing all node alignments, the alignment score is calculated as in Equation 5.2. If the score is below the threshold, the steps 2-4 are executed again. Our evaluation results in Section 5.4 show that the attack graph is usually found within the first few iterations.

5.4 Evaluation

We evaluate POIROT's efficacy and reliability in three different experiments. In the first experiment, we use a set of TC program red-team vs. blue-team adversarial engagement scenarios which are set up in an isolated network simulating an enterprise network. In the second experiment, we further test POIROT on real-world incidents whose natural language descriptions are publicly available on the internet. To reproduce the attacks described in the public threat reports, we obtained and executed their binary samples in a controlled environment and collected kernel audit logs from which we build the provenance graph. In the third experiment, we evaluate POIROT's robustness against false signals in an attack-free dataset.

In all the experiments, we set the value of AC_{thr} to 3 (and thus a threshold of $\frac{1}{3}$). This choice is validated in Section 5.4.3. We note, however, that one can configure POIROT with higher or lower values depending on the confidence about the system's protection mechanisms or the effort cyber-analysts are willing to spend to check the alarms. In fact, the value of AC_{thr} influences the number of false positives and potential false negatives. A higher AC_{thr} will increase the number of false positives while a lower AC_{thr} will reduce it. On the other hand, a higher value of AC_{thr} may detect sophisticated attacks, with multiple initial entry points, while a smaller value may miss them. After finding alignment with a score bypassing the threshold, we manually analyzed all the matched attack subgraphs to confirm that they were correctly pinpointing the actual attacks present in the query graphs.

5.4.1 Evaluation on the TC Dataset

TABLE XVIII

Characteristics of Query Graphs.

Scenario	**subjects** $\in \lvert V(G_q)\rvert$	**objects** $\in \lvert V(G_q)\rvert$	$\lvert E(G_q)\rvert$	$\lvert F(G_q)\rvert$
BSD-1	4	9	19	81
BSD-2	1	7	10	32
BSD-3	3	18	34	159
BSD-4	2	8	13	43
Win-1	13	8	26	149
Win-2	1	13	19	94
Linux-1	2	9	19	62
Linux-2	5	12	24	112
Linux-3	2	8	22	48
Linux-4	4	11	22	96

This experiment was conducted on a dataset released by the TC program, generated during a red-team vs. blue-team adversarial engagement in April 2018 [28]. In the engagement, different services were set up, including a web server, an SSH server, an email server, and an SMB server. An extensive amount of benign activities was simulated, including system administration tasks, web browsing to many web sites, downloading, compiling, and installing multiple tools. The red-team relies on threat descriptions to execute these attacks. We obtained these threat descriptions and used them to extract a query graph for each scenario (summary shown in Table XVIII).

In total, we evaluated Poirot on ten attack scenarios including four on BSD, two on Windows, and four on Linux. Due to space restrictions, we are not able to show all the query graphs; however, their characteristics are described in Table XVIII, where subjects indicate processes, and objects indicate files, memory objects, and sockets. BSD-1-4 pertain to attacks conducted on a FreeBSD 11.0 (64-bit) web-server which was running a back-doored version of Nginx. Win-1&2 pertain to attacks conducted on a host machine running Windows 7 Pro (64-bit). The Win-1 scenario contains a phishing email with a malicious Excel macro attachment, while the Win-2 scenario contains exploitation of a vulnerable version of the Firefox browser. Linux1&2 and Linux3&4 pertain to attacks conducted on hosts running Ubuntu 12.04 (64-bit) and Ubuntu 14.04 (64-bit), respectively. Linux1&3 contain in-memory browser exploits, while Linux2&4 involve a user who is using a malicious browser extension.

Alignment Score. As discussed in Section 5.3.2, Poirot iteratively repeats the node alignment procedure starting from the seed nodes with fewer candidates. Figure 21 shows the number of candidate aligned nodes for each node of G_q. Most of the nodes of G_q have less than ten candidate nodes in G_p, while there are also nodes with thousands of candidate nodes. These nodes, which appear thousands of times, are usually ubiquitous processes and files routinely accessed by benign activities, such as Firefox or Thunderbird. We remind the reader that our

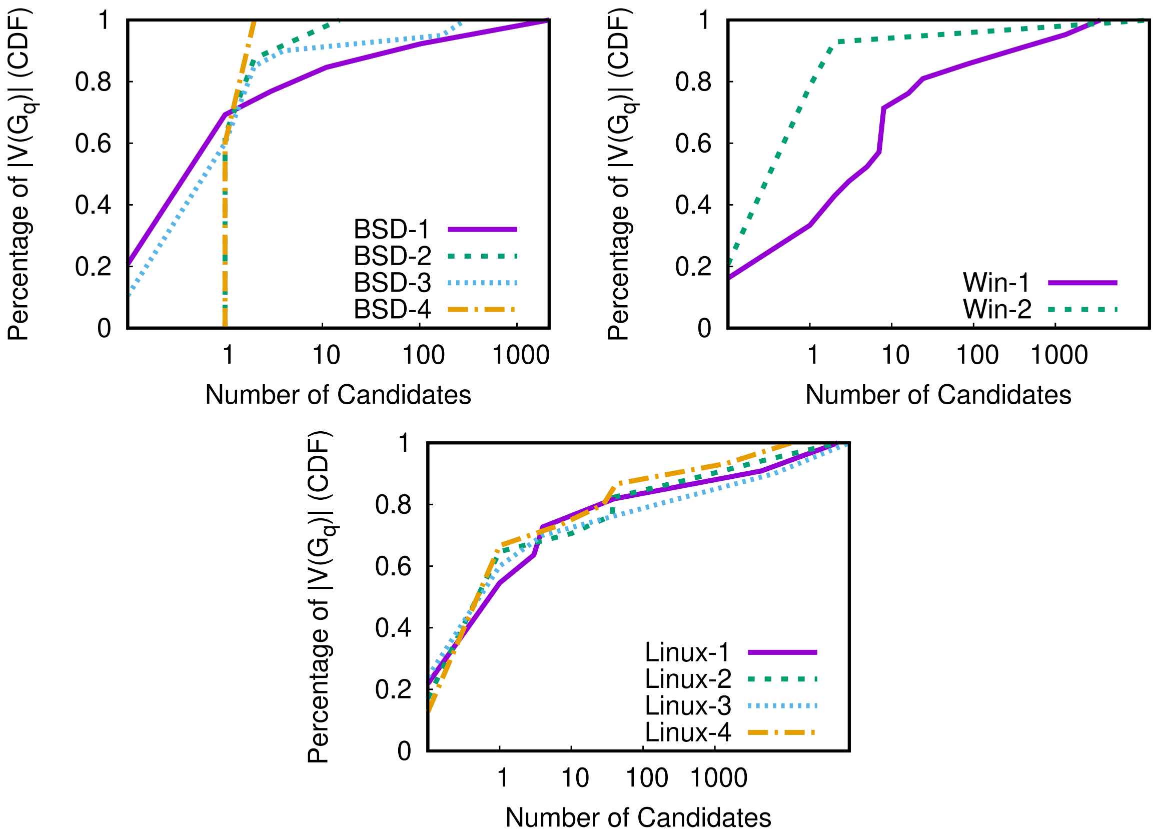


Figure 21. Cumulative Distribution Function (CDF) of number of candidates in $|G_p|$ for each node of $|G_q|$. From left to right: BSD, Windows, and Linux Scenarios.

seed nodes are chosen first from the nodes with fewer alignments. In each iteration, an alignment is constructed, and its alignment score is compared with the threshold value, which is set to $\frac{1}{3}$.

Table XIX shows POIROT's matching results for each TC scenario after producing an alignment of the query graphs with the corresponding provenance graphs. We stop the search after the first alignment that surpasses the threshold value. The second and third columns of Table XIX show the number of iterations of the steps 2-4 presented in Section 5.3.2 and the actual score obtained for the first alignment that bypasses the threshold value. In 9 out of 10 scenarios, an alignment bypassing the threshold value was found in the first iteration. In one case, the exact matching of G_q could be found in G_p (see BSD-4).

The fourth column of Table XIX shows the maximum alignment score among the 20 alignments constructed by iterating steps 2-4 of our search algorithm 20 times while the last column shows the earliest iteration-number that resulted in the maximum value. As can be seen, on average, our search converges quickly to a perfect solution. In 7 out of 10 scenarios, the maximum alignment score is calculated in the first iteration, while in the other 3, the maximum alignment scores are calculated in the fourth or fifth iterations. The latter is due to slight differences between the attack reports and the red team's implementation of the attacks, which result in information flows and causal dependencies that differ slightly between the query graph and the provenance graph. As an example, in Figure 22, we show the query graph and its

TABLE XIX

POIROT's Graph Alignment Scores.

Scenario	Earliest iteration bypassing threshold	Earliest score bypassing threshold	Max score in 20 iterations	Earliest iteration resulting Max score
BSD-1	1	0.45	0.64	5
BSD-2	1	0.81	0.81	1
BSD-3	1	0.89	0.89	1
BSD-4	1	1	1	1
Win-1	1	0.63	0.63	1
Win-2	1	0.47	0.63	4
Linux-1	1	0.58	0.58	1
Linux-2	2	0.55	0.71	5
Linux-3	1	0.54	0.54	1
Linux-4	1	0.87	0.87	1

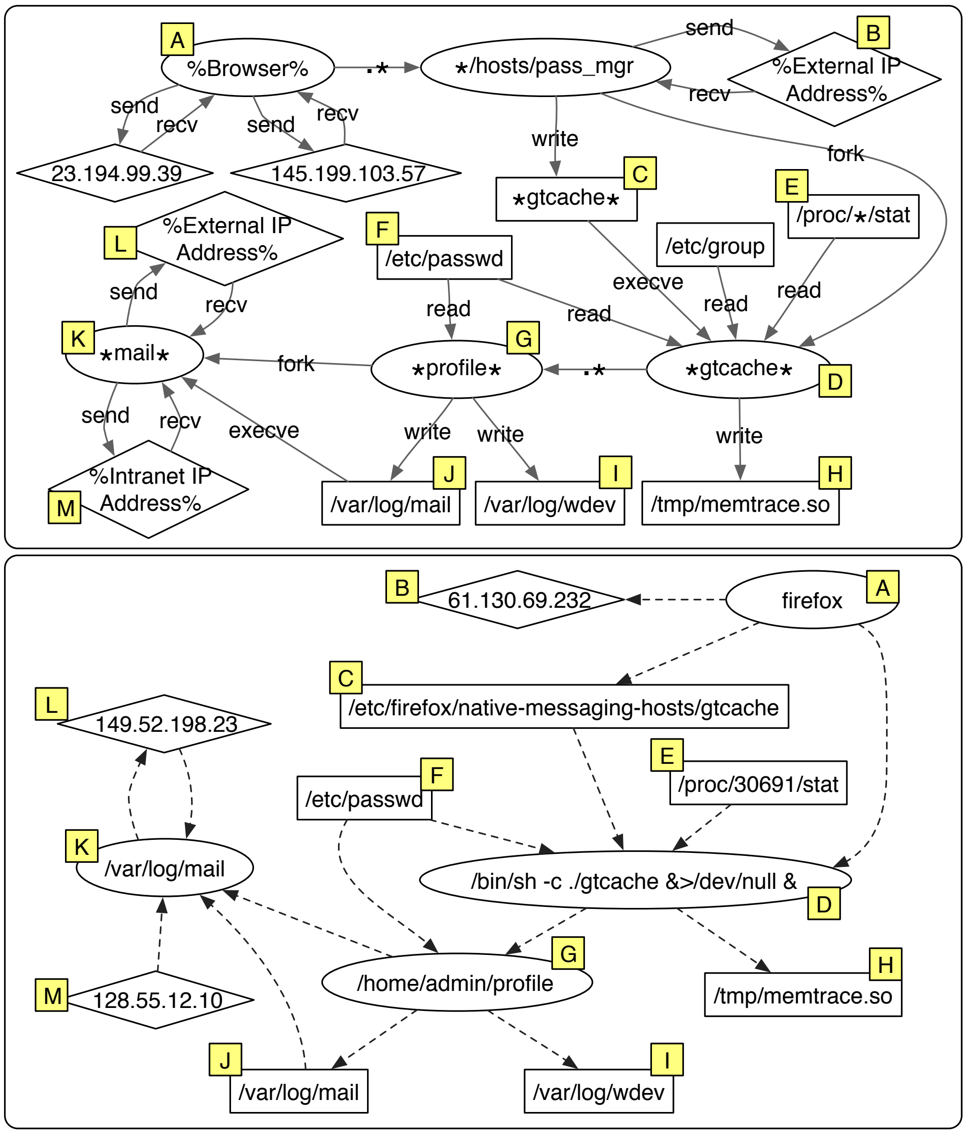


Figure 22. Query Graph of Scenario: Linux-2 (on the top) and its Detected Alignment (on the bottom).

aligned subgraph for the Linux-2 scenario. In this scenario, the attacker exploits Firefox via a malicious password manager browser extension, to implant an executable to disk. Then, the attacker runs the dropped executable to exfiltrate some confidential information and perform a port scan of known hosts on the target network. We tag the aligned nodes in each graph with the same letter label. Some nodes on the query graph are not aligned with any nodes in the provenance graph. This reduces the score of the graph alignment to a value that is less than 1. Although G_q largely overlaps with a subgraph in G_p, some nodes have no alignment, and some information flows and causal dependencies do not appear in the provenance graph. The percentage of these nodes is small, however. As long as the reports are mainly matching the actual attack activities, our approach will not suffer from this.

5.4.2 Evaluation on Public Attacks

In this section, we describe the evaluation of POIROT on attacks performed by real-world malware families and compare its effectiveness with that of other similar tools. We show the results of this evaluation in Table XX. The names of these malware families, the CTI reports we used as descriptions of their behavior, and the year in which the report is published are shown in the first three columns.

Mutation Detection Evaluation. As mentioned earlier, a common practice among attackers is that of mutating malware to evade detection or to add more features to it. Therefore, a CTI report may describe the behavior of a different version of the malware that is actually present in the system, and it is vital for a threat hunting tool to be able to detect different mutations of a malware sample. To this end, we execute several real-world malware families, containing

TABLE XX

Malware reports. In the Detection Results, B=Behavior, PE=PE-Sieve, F=File Name, H=Hash, P=Process Name, R=Registry, S=Windows Security Event.

Malware Name	**Report Source**	**Year**	**Reported Samples**	**Analyzed Malware MD5**	**Sample Relation**	**Isolated IOCs**	**Detection Results**			
							RedLine	**Loki**	**Splunk**	**POIROT**
njRAT	Fidelis [73]	2013	30	2013385034e5c8df-bbe47958fd821ca0	different	153	F+H	F+H	P	B (score=0.86)
DeputyDog	FireEye [64]	2013	8	8aba4b5184072f2a-50cbc5ecfe326701	subset	21	F×2 +H+R	F×2 +H	P+R	B (score=0.71)
Uroburos	Gdata [74]	2014	4	51e7e58a1e654b6e-586fe36e10c67a73	subset	26	F+H	F+H	R	B (score=0.76)
Carbanak	Kaspersky [75]	2015	109	1e47e12d11580e93-5878b0ed78d2294f	different	230	-	PE	S	B (score=0.68)
DustySky	Clearsky [76]	2016	79	0756357497c2cd7f-41ed6a6d4403b395	different	250	-	-	-	B (score=1.00)
OceanLotus	Eset [77]	2018	9	d592b06f9d112c86-50091166c19ea05a	subset	117	F+R	F+PE	P+R	B (score=0.65)
HawkEye	Fortinet [78]	2019	3	666a200148559e4a-83fabb7a1bf655ac	different	3	-	PE	-	B (score=0.62)

different mutated versions of the same malware, in a controlled environment. The fourth column of Table XX, shows the number of malware samples with different hash values belonging to the family mentioned in the corresponding CTI report. We note that the reports describe the behavior of only a few samples. The fifth column of Table XX shows our selected sample's hash value, while the sixth column shows the relation between our selected sample and the ones the CTI report is based on. For instance, the reports of DeputyDog, Uroburos, and OceanLotus cover different activities performed by a set of different samples, and our selected sample is one of them. We have aggregated all those activities in one query graph. For the other test cases, the sample we have executed is different from the ones that the report is based on, which could be considered as detecting a mutated malware. njRAT and DustySky explicitly mention their analyzed sample, which are different from the one we have chosen. The Carbanak report mentions 109 samples, from which we have randomly selected one. Finally, the sample of

TABLE XXI: Node labels of the query graphs in Figure 23 and their alignments.

Malware	Node	Label	Aligned Node Label
Carbanak	A	%Mail Application%	Thunderbird
	B	∗.%exe%	invitation.exe
	C	∗	invitation
	D	%system32%\svchost	C:\Windows\SysWOW64\svchost.exe:WofCompressedData
	E	svchost	svchost
	F	∗Sys$	None
	G	%COMMON_APPDATA%\Mozilla\∗.%exe%	C:\ProgramData\Mozilla\BwgWXFhfbVpfWgJfBg.bin
	H	[HKCU]\Software\Microsoft\Windows\ CurrentVersion \Internet Settings	[HKCU]\Software\Microsoft\Windows\CurrentVersion \Internet Settings
	I	%AppData%\Mozilla\Firefox\∗\prefs.js	C:\Users\test_user\AppData\Roaming\Mozilla\Firefox\ Profiles\ddl1t72n.default\prefs.js
	J	%External IP address%	None
Uroburos	A	∗	contract.exe
	B	%APPDATA%\Microsoft\credprov.tlb	C:\Users\test_user\AppData\Roaming\Microsoft\credprov.tlb
	C	%APPDATA%\Microsoft\shdocvw.tlb	C:\Users\test_user\AppData\Roaming\Microsoft\shdocvw.tlb
	D	rundll32	rundll32
	E	[HKCU]\Software\Classes\CLSID\42aedc87-2188-41fd-b9a3-0c966feabec1\	[HKCU]\Software\Classes\CLSID\42aedc87-2188-41fd-b9a3-0c966feabec1\
	F	∗\winview.ocx	None
	G	∗\mskfp32.ocx	None
	H	∗\msvcrtd.tlb	None
	I	%APPDATA%\Microsoft\oleaut32.dll	C:\Users\test_user\AppData\Roaming\Microsoft\oleaut32.dll
	J	%APPDATA%\Microsoft\oleaut32.tlb	C:\Users\test_user\AppData\Roaming\Microsoft\oleaut32.tlb
	K	%APPDATA%\Microsoft\libadcodec.dll	C:\Users\test_user\AppData\Roaming\Microsoft\libadcodec.dll
	L	%APPDATA%\Microsoft\libadcodec.tlb	C:\Users\test_user\AppData\Roaming\Microsoft\libadcodec.tlb
DustySky	A	∗.%exe%	News.docx.exe
	B	∗	News
	C	%Microsoft Word%	C:\Program Files\Microsoft Office\Office12\Winword.exe
	D	∗\vboxmrxnp.dll	C:\WINDOWS\vboxmrxnp.dlls
	E	∗\vmbusres.dll	C:\WINDOWS\vmbusres.dlls
	F	∗\vmGuestlib.dll	C:\WINDOWS\SysWOW64\vmGuestLib.dll
	G	%TEMP%\∗.%exe%	C:\Users\test_user\AppData\Local\Temp \1371372533114561232114361100131187183149253.exe
	H	∗	1371372533114561232114361100131187183149253
	I	%TEMP%\temps	C:\Users\test_user\AppData\Local\Temp\temps
	A	∗.%exe%	Chi tiet don khieu nai gui saigontel.exe

Continued on next page

Table XXI – *Continued from previous page*

Malware	Node	Label	Aligned Node Label
OceanLotus	B	∗	Chi tiet don khieu nai gui saigontel
	C	%temp%\∗	C:\Users\test_user\AppData\Local\Temp\tmp.docx
	D	%temp%\[0-9].tmp.%exe%	None
	E	%Microsoft Word%	C:\Program Files\Microsoft Office\Office12\Winword.exe
	F	∗\rastlsc.%exe%	C:\Program Files (x86)\Symantec\Officewordtask\rastlsc.exe
	G	∗\rastls.dll	C:\Program Files (x86)\Symantec\Officewordtask\rastls.dll
	H	∗\(Sylog.bin|OUTLFLTR.DAT)	C:\Program Files (x86)\Symantec\ Officewordtask\OUTLFLTR.DAT
	I	rastlsc	rastlsc
	J	\SOFTWARE\Classes\AppX∗	None
	K	∗\HTTPProv.dll	None
	L	SOFTWARE\Classes\CLSID\{E3517E26-8E93-458D-A6DF-8030BC80528B}	SOFTWARE\Classes\CLSID\{E3517E26-8E93-458D-A6DF-8030BC80528B}
	M	%External IP address%	None
njRAT	A	∗	Authorization
	B	∗.exe.config	C:\Users\test_user\Desktop\Authorization.exe.config
	C	∗.tmp	C:\Users\test_user\AppData\Roaming\ja33kk.exe.tmp
	D	C:\WINDOWS\Prefetch\∗.EXE-∗.pf	C:\Windows\Prefetch\AUTHORIZATION.EXE-69AD75AA.pf
	E	%APPDATA%\∗	C:\Users\test_user\AppData\Roaming\ja33kk.exe
	F	∗	ja33kk
	G	C:\WINDOWS\Prefetch\∗.EXE-∗.pf	C:\Windows\Prefetch\JA33KK.EXE-7FA5E873.pf
	H	%USER_PROFILE%\Start Menu\ Programs\Startup\∗	C:\Users\test_user\AppData\Roaming\Microsoft\Windows\ Start Menu\Programs\Startup\- 9758a8dfbe15a00f55a11c8306f80da1.exe
	I	netsh	netsh
	J	C:\WINDOWS\Prefetch\NETSH.EXE-∗.pf	C:\Windows\Prefetch\NETSH.EXE-CD959116.pf
	K	[HKCU]\Software\Microsoft\Windows\ CurrentVersion\Run\	[HKCU]\Software\Microsoft\Windows\CurrentVersion\Run\
	L	[HKLM]\Software\Microsoft\Windows\ CurrentVersion\Run\	[HKLM]\Software\Microsoft\Windows\CurrentVersion\Run\
	M	[HKLM]\SYSTEM\CurrentControlSet\ Services\SharedAccess\Parameters\ FirewallPolicy\StandardProfile\Authorized-Applications\List\APPDATA\	None
	N	%External IP address%	None
	A	∗.%Compressed%	PROFORMA INVOICE _20190423072201 pdf.bin.zip
	B	%Unachiever%	WinRAR.exe

Continued on next page

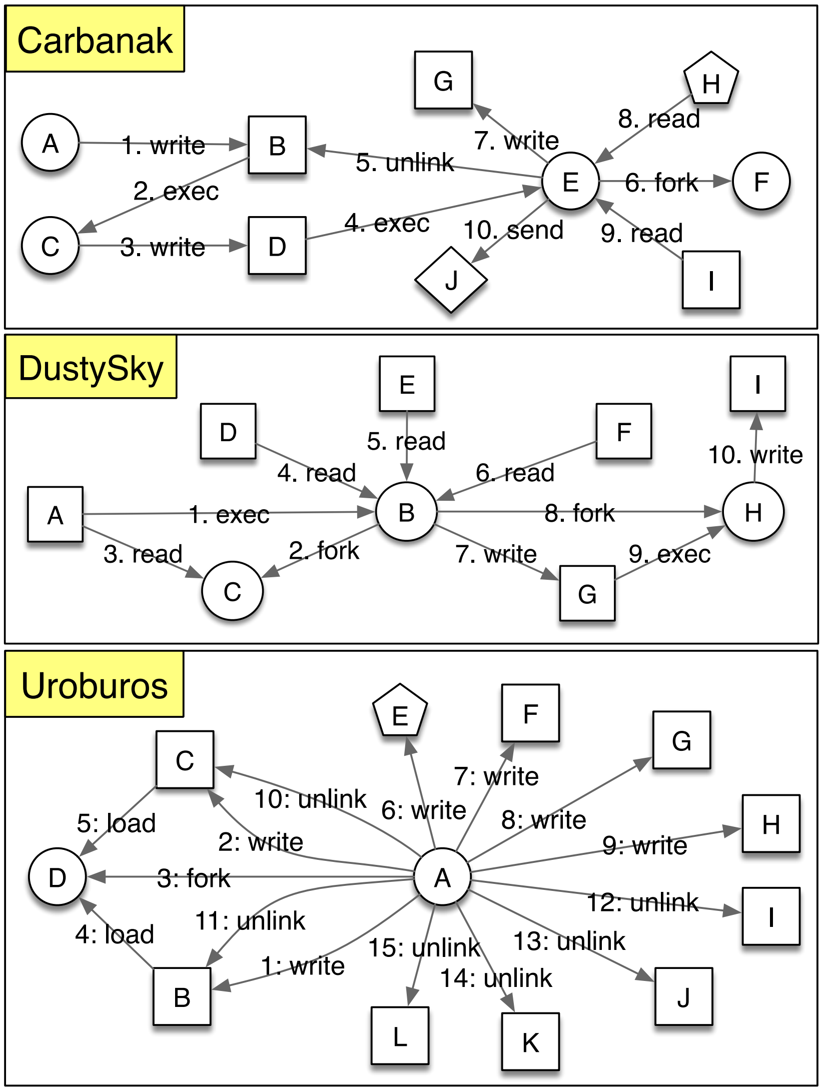
Table XXI – *Continued from previous page*

Malware	**Node**	**Label**	**Aligned Node Label**
HawkEye	C	*.%exe%	C:\Users\test_user\Desktop\PROFORMA INVOICE _20190423072201 pdf.bin
	D	*	PROFORMA INVOICE _20190423072201 pdf
	E	RegAsm	RegAsm
	F	vbc	vbc (PID_1)
	G	vbc	vbc (PID_2)
	H_1	*Opera*	C:\Users\test_user\AppData\Roaming\Opera\Opera7\profile\wand.dat
	H_2	*Chrome*	C:\Users\test_user\AppData\Local\Google\Chrome\User Data\Default\Login Data
	H_3	*Chromium*	C:\Users\test_user\AppData\Local\Chromium\User Data
	H_4	*Chrome SxS*	C:\Users\test_user\AppData\Local\Google\Chrome SxS\User Data
	H_5	*Thunderbird*	C:\Users\test_user\AppData\Roaming\Thunderbird\Profiles
	H_6	*SeaMonkey*	C:\Users\test_user\AppData\Roaming\Mozilla\SeaMonkey\Profiles
	H_7	*SunBird*	None
	H_8	*IE*	C:\Users\test_user\AppData\Local\Microsoft\Windows\History\History.IE5
	H_9	*Safari*	None
	H_{10}	*Firefox*	C:\Users\test_user\AppData\Roaming\Mozilla\Firefox\profiles.ini
	H_{11}	*Yandex*	C:\Users\test_user\AppData\Local\Yandex\YandexBrowser\User Data\Default\Login Data
	H_{12}	*Vivaldi*	C:\Users\test_user\AppData\Local\Vivaldi\User Data\Default\Login Data
	I_1	*Yahoo*	[HKLM]\Software\Yahoo\Pager
	I_2	*GroupMail*	None
	I_3	*Thunderbird*	C:\Users\test_user\AppData\AppData\Roaming\Thunderbird\Profiles
	I_4	*MSNMessenger*	[HKLM]\Software\Microsoft\MSNMessenger
	I_5	*Windows Mail*	C:\Users\test_user\AppData\Local\Microsoft\Windows Mail
	I_6	*IncrediMail*	[HKLM]\Software\WOW6432Node\IncrediMail\Identities
	I_7	*Outlook*	[HKLM]\Software\Microsoft\Office\16.0\Outlook\Profiles
	I_8	*Eudora*	[HKLM]\Software\Qualcomm\Eudora\CommandLine
	J	%temp%*.tmp	C:\Users\test_user\AppData\Local\Temp\tmp8FC3.tmp
	K	%temp%*.tmp	C:\Users\test_user\AppData\Local\Temp\tmp8BAB.tmp
	L	http[s]:\\whatismyipaddress.com*	None

Continued on next page

Table XXI – *Continued from previous page*

Malware	**Node**	**Label**	**Aligned Node Label**
	M	%External IP address%	None
DeputyDog (Figure 19)	A	∗.%exe%	C:\Users\test_user\Desktop\img20130823.jpg.exe
	B	∗	img20130823
	C	%APPDATA%\∗	C:\ProgramData\28542CC0.dll
	D	[HKCU]\Software\Microsoft\Windows\ CurrentVersion\ Run\	None
	E	%External IP address%	180.150.228.102



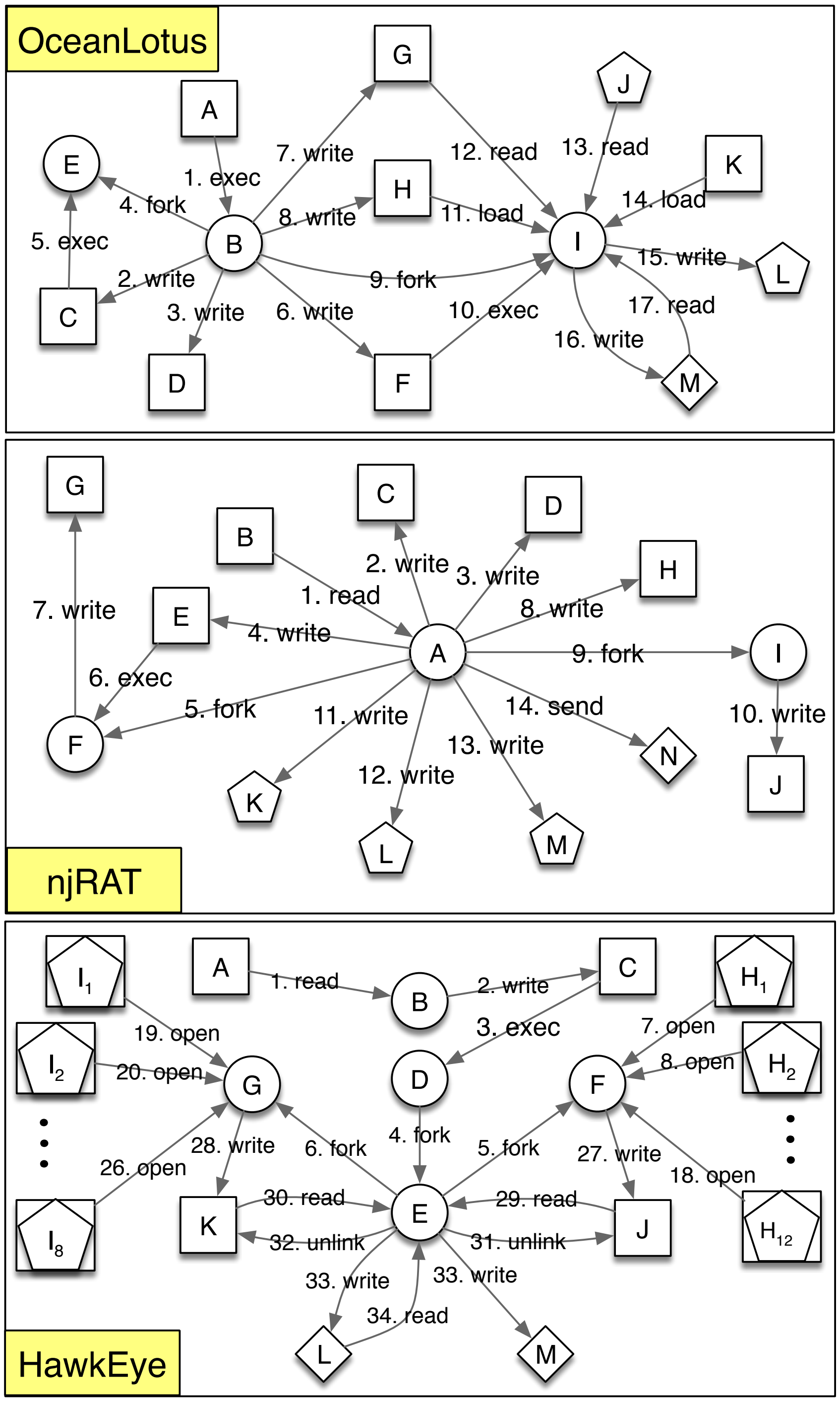


Figure 23. Query Graphs of Carbanak, DustySky, Uroburos, OceanLotus, njRAT, and HawkEye malware extracted from their CTI reports.

HawkEye malware is selected from an external source and is not among the samples mentioned in the report.

Comparison with Existing Tools. We compare POIROT with the results of three other tools, namely RedLine [12], Loki [11], and Splunk [15]. The input to these tools is extracted from the same report we extract the query graphs and contains IOCs in different types such as hash values, process names, file names, and registries. We have transformed these IOCs to the accepted format of each tool (e.g., RedLine accepts input in OpenIOC format [45]). The number of IOCs submitted to Redline, Loki, and Splunk are shown in column-7, while the query graphs submitted to POIROT are shown in Figure 23. A detailed explanation of these query graphs demonstrating how they are constructed can be found in Appendix C. The correspondence between node labels in the query graphs and their actual names is represented in the second and third columns of Table XXI, while the alignments produced by POIROT are shown in the last column.

As shown in the extracted query graphs, the design of POIROT 's search method, which is based on the information flow and causal dependencies, makes us capable to include benign nodes (nodes C, D, E, and F in DustySky) or attack nodes with exact same names of benign entities (node E in Carbanak) in the query graph. However, these entity names could not be defined as an IOC in the other tested tools as will lead to many false positive alarms. As Redline, Loki, and Splunk look for each IOC in isolation, they expect IOCs as input that are always malicious regardless of their connection with other entities. To this end, we do a preliminary search for each isolated IOC in a clean system and make sure that we have only extracted IOCs that have no match in a clean system. As a result, for some test cases, like HawkEye, although the behavior graph is rich, there are not so many isolated IOCs except a few hash values that could be defined. This highlights the importance of the dependencies between IOCs, which is the foundation of POIROT's search algorithm, and is not considered by other tools.

Detection Results. The last four columns of Table XX contain the detection results, which show how each tool could detect the tested samples. Keywords B, F, H, P, and R represent detection based on the behavior, file name, hash value, process name, and registry, respectively. In addition, some of the tested tools feature other methods to detect anomalies, injection, or other security incidents. Among these, we encountered some alarms from Windows Security Mitigation and PE-Sieve [79], which are represented by keywords S and PE, respectively. While for POIROT, a score is shown which shows the goodness of the overall alignment of each query graph, for the other tools, $\times N$ indicates the number of hits when there has been more than one hit for a specific type of IOC.

As shown in Table XX, for all the test cases, POIROT has found an alignment that bypasses the threshold value of $\frac{1}{3}$. After running the search algorithm, in most of the cases, POIROT found a node alignment for only a subset of the entities in the query graph, except for DustySky, where POIROT found a node alignment for every entity. The information flows and causal dependencies that appear among the aligned nodes are often the same as the query graph with some exceptions. For example, in contrast to how it appears in the query graph of njRAT, where node A generates most of the actions, in our experiment, node F generated most of the actions, such as the write event to nodes C, G, K, L, and the fork event of node I. However, since there is a path from node A to node F in the query graph, Poirot was able to find these alignments and measure a high alignment score.

The samples of njRAT, DeputyDog, Uroburous, and OceanLotus are also detected by all the other tools, as these samples use unique names or hash values that are available in the threat intelligence and could be attributed to those malwares. For the other three test cases, none of the isolated IOCs could be detected because of different reasons such as malware mutations, using random names in each run (nodes J and K in HawkEye query graph), and using legitimate libraries or their similar names. Nevertheless, Splunk found an ETW event related to the Carbanak sample, which is generated when Windows Security Mitigation service has blocked svchost from generating dynamic code. Loki's PE-Sieve has also detected some attempts of code implants which have resulted in raising some warning signal and not an alert. PE-Sieve detects all modifications done to a process even though they may not necessarily be malicious. As such modifications happen regularly in many benign scenarios, PE-Sieve detections are considered as warning signals that need further investigations.

Conclusions. Our analysis results show that other tools usually perform well when the sample is a subset of the ones the report is written based on. This situation is similar to when there is no mutations, and therefore, there are unique hash values or names that could be used as signature of a malware. For example, DeputyDog sample drops many files with unique names and hash values that do not appear in a benign system, and finding them is a strong indication of this malware. However, its query graph (Figure 19) is not very rich, and Poirot has not been able to correlate the modified registry (node D) with the rest of the aligned nodes. Although the calculated score is still higher than the threshold, but the other tools might perform better when the malware is using well-known IOCs that are strong enough to indicate an attack in isolation.

On the contrary, when the chosen sample is different from the samples analyzed by the report, which is similar to the case that malware is mutated, other tools usually are not able to find the attacks. In such situations, POIROT has a better chance to detect the attack as the behavior often remains constant among the mutations.

5.4.3 Evaluation on Benign Datasets

To stress-test POIROT on false positives, we used the benign dataset generated as part of the adversarial engagement in the TC program and four machines (a client, a SSH server, a mail server and a web server) we monitored for one month. Collectively, these datasets contained over seven months worth of benign audit records and billions of audit records on Windows, Linux, and FreeBSD. During this time, multiple users used these systems and typical attack-free actions were conducted including web browsing, installing security updates (including kernel updates), virus scanning, taking backups, and software uninstalls.

After collecting the logs, we run POIROT to construct the provenance graph, and then search for all the query graphs we have extracted from the TC reports and the public malware reports. We try up to 20 iterations starting from different seed node selections per each query graph per each provenance graph and select the highest score. Note that although these logs are attack-free, they share many nodes and events with our query graphs, such as confidential files, critical system files, file editing tools, or processes related to web browsing/hosting, and

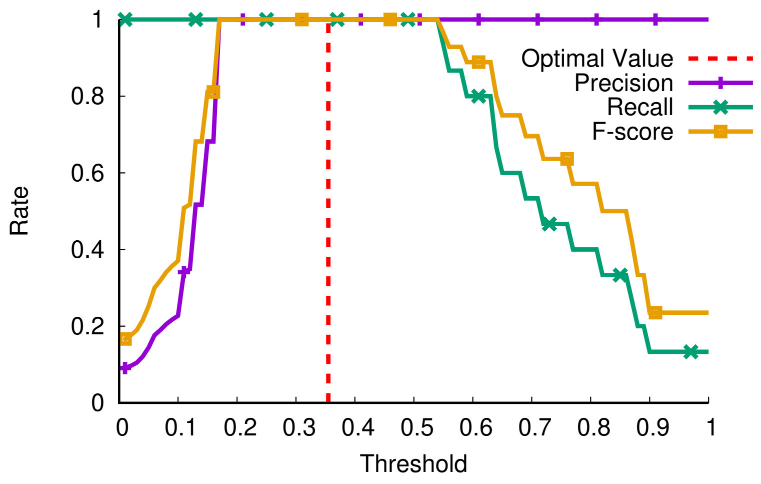


Figure 24. Selecting the Optimal Threshold Value.

email clients, all of which were accessed during the benign data collection period. However, even in cases where similar flows appear by chance, the *influence score* prunes away many of these flows. Consequently, the graph alignment score POIROT calculates among all the benign datasets is at most equal to 0.16, well below the threshold.

Validating the Threshold Value. The selection of the threshold value is critical to avoid false signals. Too low a threshold could result in premature matching (false positives) while too high a threshold could lead to missing reasonable matches (false negatives). Thus, there is a trade-off in choosing an optimal threshold value. To determine the optimal threshold value, we measured the *F-score* using varying threshold values, as shown in Figure 24. This analysis is done based on the highest alignment score calculated in 20 iterations of POIROT's search algorithm for all the attack and benign scenarios we have evaluated. As it is shown, the highest

F-score value is achieved when the threshold is at the interval [0.17, 0.54], which is the range in which all attack subgraphs are correctly found, and no alarm is raised for benign datasets. The middle of this interval, i.e., 0.35, maximizes the margin between attack and benign scores, and choosing this value as the optimal threshold minimizes the classification errors,. Therefore, we set the AC_{thr} to 3 which results in $\frac{1}{AC_{thr}} = \frac{1}{3}$ which is close to the optimal value.

5.4.4 Efficiency

The overheads and search times for the different tools we used are shown in Table XXII. Redline and Loki are offline tools, searching for artifacts that are left by the attacker on the system, while Splunk and POIROT are online tools, searching based on system events collected during runtime. Hence, Redline and Loki have no runtime overhead due to audit log collection. The runtime overheads of Splunk and POIROT due to log collection are measured using Apache benchmark [80], which measures web server responsiveness, JetStream [81], which measures browser execution times, and HDTune [82], which measures heavy hard drive transactions. As shown in Table XXII, both tools have shown negligible runtime overhead, while the runtime

TABLE XXII

Efficiency Comparison with Related Systems.

Detection Method	**Type**	**Runtime Overhead**			**Search Time (min)**
		Apache [80]	**JetStream [81]**	**HDTune [82]**	
Redline	offline	-	-	-	124
Loki	offline	-	-	-	215
Splunk	online	3.70%	2.94%	4.37%	¡ 1
POIROT	online	0.82%	1.86%	0.64%	¡ 1

of Splunk can be further improved by setting it up in a distributed setting and offloading the data indexing task to another server.

The last column of Table XXII shows the time it took searching for IOCs per each tool. The search time of offline tools highly depends on the number of running processes and volume of occupied disk space, which was 500 GB in our case. On the other hand, the search time of online methods highly depends on the log size, type and number of activities represented by the logs. As our experiments with real-world malware samples were running in a controlled environment without many background benign activities and Internet connection, both Splunk and POIROT spend less than one minute to search for all the IOCs mentioned in Table XX. In the following, we perform an in-depth analysis of POIROT 's efficiency on the TC scenarios, which overall contain over a month worth of log data with combined attack and benign activities. The analysis is done on an 8-core CPU with a 2.5GHz speed each and a 150GB of RAM.

Audit Logs Consumption. In Table XXIII, the second column shows the initial size of the logs on disk, the third column represents the time it takes to consume all audits log events

TABLE XXIII

Statistics of logs, Consumption and Search Times.

Scenario	**Size on Disk (Uncompressed)**	**Consumption time**	**Occupied Memory**	**Log Duration**	**sub** $\in$ $\lvert V(G_p)\rvert$	**obj** $\in$ $\lvert V(G_p)\rvert$	$\lvert E(G_p)\rvert$	**Search Time (s)**
BSD-1	3022 MB	0h-34m-59s	867 MB	03d-18h-01m	110.66 K	1.48 M	7.53 M	3.28
BSD-2	4808 MB	0h-58m-05s	1240 MB	05d-01h-15m	213.10 K	2.25 M	12.66 M	0.04
BSD-3	1828 MB	0h-21m-31s	638 MB	02d-00h-59m	84.39 K	897.63 K	4.65 M	26.09
BSD-4								1.47
Win-1	54.57 GB	4h-58m-30s	3790 MB	08d-13h-35m	1.04 M	2.38 M	70.82 M	125.26
Win-2								46.02
Linux-1	9436 MB	1h-26m-37s	4444 MB	03d-04h-20m	324.68 K	30.33 M	51.98 M	1279.32
Linux-2								1170.86
Linux-3	131.1 GB	2h-30m-37s	21.2 GB	10d-15h-52m	374.71 K	5.32 M	69.89 M	385.16
Linux-4	4952 MB	0h-04m-00s	1095 MB	00d-07h-13m	35.81 K	859.03 K	13.06 M	20.72

from disk for building the provenance graph in memory. This time is measured as the wall-clock time and varies depending on the size of each audit log and the structure of audits logs generated in each platform (BSD, Windows, Linux). The fourth column shows the total memory consumption by each provenance graph. Comparing the size on disk versus memory, we notice that we have an average compression of 1:4 (25%) via a compact in-memory provenance graph representation based on [1]. However, if memory is a concern, it is still possible to achieve better compression using additional techniques proposed in this area [36, 83, 84]. The fifth column shows the duration during which the logs were collected while columns 6, 7, and 8 show the total number of subjects (i.e. processes), objects, and events in the provenance graph that is built from the logs, respectively. We note that the *query graphs* are on average 209K times smaller than the provenance graph for these scenarios. Nevertheless, POIROT is still able to find the exact embedding of G_q in G_p very fast, as shown in the last column. We note that some scenarios are joined (e.g., Win-1&2) because they were executed concurrently on the same machines.

Graph Analytics. In the last column of Table XXIII, we show the runtime of graph analytics for POIROT's search algorithm. These times are measured from the moment a search query is submitted until we find a similar graph in G_p with an alignment score that surpasses the threshold. Therefore, for Linux-2, the time includes the sum of the times for two iterations. The main bottleneck is on the graph search expansion (Step 3), and the time POIROT spends on graph search depends on several factors. Obviously, the sizes of both query and provenance graph are proportional to the runtime. However, we notice that the node names in G_q and the shape of this graph have a more significant effect. In particular, when there are nodes with many candidate alignments, there is a higher chance to reverse the direction multiple times and runtime increases accordingly.

5.5 Summary

Poirot formulates cyber threat hunting as a graph pattern matching problem to reliably detect known cyber attacks. Poirot is based on an efficient alignment algorithm to find an embedding of a graph representing the threat behavior in the provenance graph of kernel audit records. We evaluate Poirot on real-world cyber attacks and on ten attack scenarios conducted by a professional red-team, over three OS platforms, with tens of millions of audit records. Poirot successfully detects all the attacks with high confidence, no false signals, and in a matter of minutes.

CHAPTER 6

RELATED WORK

This chapter includes excerpts and figures from material that is published in [1–3].

In this chapter, we discuss the closely related work to the approaches presented in this dissertation. Specifically, we cover the related research in the areas of provenance graph, intrusion detection, alert correlation, query processing systems, behavior discovery, and graph pattern matching.

6.1 Provenance Graph

Several logging and provenance tracking systems have been built to monitor the activities of a system [38, 39, 85–89] and build *provenance graphs*. Among these, *Backtracker* [8, 9] is one of the first works that used dependence graphs to trace back to the root causes of intrusions. These graphs are built by correlating events collected by a logging system and by determining the causality among system entities, to help in forensic analysis after an attack is detected.

Sleuth improves on the techniques of Backtracker in two important ways. First, Backtracker was meant to operate in a forensic setting, whereas our analysis and data representation techniques are designed towards real-time detection. Setting aside hardware comparisons, we note that Bactracker took 3 hours for analyzing audit data from a 24-hour period, whereas Sleuth was able to process 358 hours of logs in a little less than 3 minutes. Secondly, Backtracker relies on alarms generated by external tools, therefore its forensic search and pruning cannot leverage the reasons that generated those alarms. In contrast, our analysis procedures leverage the results from our principled tag-based detection methods and therefore are inherently more precise. For example, if an attack deliberately writes into a well-known log file, Backtracker's search heuristics may remove the log file from the final graph, whereas our tag-based analysis will prevent that node from being pruned away.

To have a more efficient and effective use of provenance graphs, several approaches have introduced compression, summarization, and log reduction techniques [36,83,84,90] to differentiate worthy events from uninformative ones and consequently reduce the storage size. Dividing processes into smaller units is one of the approaches to add more granularity into the provenance graphs, and to this end, researchers have utilized different methods, such as dynamic binary analysis [25,40], source code annotation [91], or modeling-based inference [4,92,93]. Additionally, record-and-replay [94,95] and parallel execution methods [96] are proposed for more precise tracking. Similar to our proposed methods, some recent studies have leveraged provenance graphs for different objectives, such as timely Causality Analysis [97], alert triage [98] and zero-day attack path identification [99].

While the majority of the aforementioned systems operate at the system call level, several other systems track information flows at finer granularities [23–25]. They typically instrument applications (e.g., using Pin [26]) to track information flows through a program. Such fine-grained tainting can provide much more precise provenance information, at the cost of higher overhead. fine-grained taint tracking not only can mitigate memory corruption vulnerabilities in low-level languages [29], but also can cover a much broader range of vulnerabilities that afflicted programs in higher-level and scripting languages [30]. It was also demonstrated to be a very powerful technique for malware analysis and automated generation of patches. Finally, it has been shown that the power of IDS techniques can be significantly enhanced using fine-grained taint [100].

6.2 Intrusion Detection

Offline *intrusion detection* using logs has been studied for a long period [101–103], and different sources of logs are considered. BotHunter [104] analyzes network traffic to detect malware infections. Opera et al. [105] leverage DNS or web proxy logs for detecting early-stage infection in an enterprise. Disclosure [106] extracts statistical features from NetFlow logs to detect botnet C&C channels. DNS logs have also been extensively used [107, 108] for detecting malicious domains.

Host-based IDS using system-call monitoring and/or audit logs has been investigated by numerous research efforts [109–112]. These approaches fall under three classes: (1) *misuse-based* [41, 42], which detect behavior associated with known attacks; (2) *anomaly-based* [103, 110, 113–119], which learn a model of benign behavior and detect deviations from it; and (3) *specification-based* [120, 121], which detect attacks based on policies specified by experts. While the techniques of the first class cannot deal with unknown attacks, those of the second class can produce many false positives. At a superficial level, the use of TTPs in Holmes can be seen as an instance of misuse detection. However, our approach goes beyond classic misuse detection [41, 42] in the use of prerequisite-consequence patterns that are matched when there exist information flow dependencies between the entities involved in the matched TTP patterns.

6.3 Alert Correlation

Network IDSs often produce myriad alerts. *Alert correlation* analyzes relationships among alerts, to help users deal with the deluge. The main approaches, often used together, are to *cluster* similar alerts, prioritize alerts, and identify causal relationships between alerts [122–126]. Hercule [49] uses community discovery techniques to correlate attack steps that may be dispersed across multiple logs. Moreover, industry uses similar approaches for building SIEMs [15–17] for alert correlation and enforcement based on logs from disparate data sources. These approaches rely on logs generated by third-party applications running in user-space. Moreover, alert correlation based on statistical features like alert timestamps does not help in precise detection of multi-stage APT attacks as they usually span a long duration.

In contrast to these approaches, Holmes builds on information flows that exist between various attack steps for the purpose of alert correlation. The use of kernel audit data in this context was first pursued in [127]. However, differently from Holmes, that work is purely misuse-based, and its focus is on using the correlation between events to detect steps of an attack that are missed by an IDS. Holmes uses the same kernel audit data but pursues a different approach based on building a main-memory dependency graph with low memory footprint, followed by the derivation of an HSG based on the high-level specification of TTPs to raise alerts, and finally correlate alerts based on the information flow between them. An additional line of work on alert correlation relies on the proximity of alerts in time [128]. Holmes, in contrast, relies on information flow and causality connections to correlate alerts and is therefore capable of detecting even attacks where the steps are executed very slowly.

6.4 Query Processing Systems

Prior works have incorporated novel optimization techniques, graph indexing, and query processing methods [129–131] to support timely attack investigations. SAQL [132] is an anomaly query engine that queries specified anomalies to identify abnormal behaviors among system events. AIQL [133] can be used as a forensic query system that has a domain-specific language for investigating attack campaigns from historical audit logs. Pasquier et al. [134] propose a query framework, called CamQuery, that supports real-time analysis on provenance graphs, to address problems such as data loss prevention, intrusion detection, and regulatory compliance. Shu et al. [135] also propose a human-assisted query system equipping threat hunters with a suite of potent new tools. These works are orthogonal to Poirot and can be used as a foundation to implement our search algorithm.

6.5 Behavior Discovery

Extracting malicious behaviors such as information flows and causal dependencies and searching for them as robust indicators have been investigated in prior works. Christodorescu et al. [136] have proposed an approach for mining malware behavior from dynamic traces of that malware's samples. Similarly, Kolbitsch et al. [52] automatically generate behavior models of malware using symbolic execution. They represent this behavior as a graph and search for it among the runtime behavior of unknown programs. On the contrary, Poirot does not rely on symbolic expressions but looks for correlations and information flows on the whole system. TGMiner [53] is a method to mine discriminative graph patterns from training audit logs and search for their existence in test data. The focus of this work is query formulation instead of pattern query processing, and the authors have used a subsequence matching solution [65] for their search, which is different from our graph pattern matching approach.

6.6 Graph Pattern Matching

Graph Pattern Matching (GPM) has proved useful in a variety of applications [66]. GPM can be defined as a search problem inside a large graph for a subgraph containing similar connections conjunctively specified in a small query graph. This problem is NP-complete in the general case [56]. Fan et. al. [67] proposed a polynomial time approach assuming that each connection in the pattern could only be mapped to a path with a predefined number of hops. Other works [68,69] have tackled the problem by using a sequence of join functions in the vector space. NeMa [70] is a neighborhood-based subgraph matching technique based on the proximity of nodes. In contrast, G-Ray and later Mage [71,72] take into account the shape of the query graph and edge attributes and are more similar to our approach, where similar information flows and causal dependencies play a crucial role. However, these approaches work based on random-walk, which is not reliable against attackers (with knowledge of the threat-hunting method) who generate fake events (as explained in 5.3.1). While our graph alignment notions are similar to these works, the graph characteristics Poirot analyzes present new challenges such as being labeled, directed, typed, in the order of millions of nodes, and constructed in an adversarial setting. Moreover, many of these related works are looking for a subgraph that contains exactly one alignment for each node and each edge of the query graph and cannot operate in a setting where there might not be an alignment for certain nodes or edges. As a result, we develop a new best-effort matching technique aimed at tackling these challenges.

CHAPTER 7

CONCLUSION AND FUTURE WORK

The causal linkage among system events plays a crucial role to make reasoning in cybersecurity tasks. In this dissertation, we proposed efficient and effective methods utilizing the causal linkage among kernel audit logs to boost cybersecurity tasks, such as APT detection, scenario reconstruction, and threat-hunting. The proposed systems leverage a platform-neutral graph representation stored in the main memory, which is constructed by processing the sequence of kernel audit logs.

In chapter 3, we presented an approach and a system called Sleuth for real-time detection of attacks and attack reconstruction from kernel audit logs. Sleuth uses a rich tag-based policy framework that make its analysis both efficient and precise. We evaluated Sleuth on large datasets from 3 major OSes under attack by an independent red team, efficiently reconstructing all the attacks with very few errors.

In chapter 4, we present Holmes, a real-time APT detection system that correlates tactics, techniques, and procedures that might be used to carry out each APT stage. Holmes generates a high-level graph that summarizes the attacker's steps in real-time. We evaluate Holmes against nine real-world APT threats and deploy it as a real-time intrusion detection tool. The results show that Holmes successfully detects APT campaigns with high precision and low false alarm rates.

Chapter 5 formulates cyber threat-hunting as a graph pattern matching problem to reliably detect known cyberattacks. The proposed system, called Poirot, is based on an efficient alignment algorithm to find an embedding of a graph representing the threat behavior in the provenance graph of kernel audit records. We evaluate Poirot on real-world cyberattacks and on ten attack scenarios conducted by a professional red-team, over three OS platforms, with tens of millions of audit records. Poirot successfully detects all the attacks with high confidence, no false signals, and in a matter of minutes.

Taken together, by proposing the aforementioned systems in this dissertation, we demonstrated that the low-level causal information inferred from kernel audit logs could be utilized to achieve robust and reliable threat detection methods that efficiently pinpoint threats and reveal the high-level picture of attacks by producing compact visual graphs of attack steps. While Sleuth, Holmes and Poirot focus on scenario reconstruction, APT detection and cyber threat-hunting, respectively, there are also other security-related tasks that might be enhanced leveraging the semantically-meaningful connections in the kernel audit logs. For example, kernel audit logs are shown to be useful for expediting forensic analysis [97], alert triage [98], and zero-day attack path identification [99]. In addition to these recent works, there are other objectives we believe can benefit from this type of analytics in future:

- Lateral movement Detection: it is a common practice among attackers to progressively move across different machines in a network to get access to high value assets. In these attacks, the attacker compromises one machines and tries to laterally compromise other machines to establish a foothold or to access more sensitive data. Detecting lateral movements is a big challenge for the current state-of-the-art systems as it seems like a legitimate communication from an insider machine.

- Similarity Search: when we detect a compromise on a machine, it is very likely that multiple machines in an organization or across multiple organizations are also compromised by the same incident. Extracting an attack pattern from the detected instance and searching for similar instances can help to enhance security and improve the current detection systems.
- Hypothesis testing: security admins often come up with different hypotheses in their daily tasks and are interested to validate them. For example, system admin notices a suspicious executable file on *Desktop* and the existence of many running *svchost* processes in the background. A hypothesis can be that the *svchost* processes have a correlation with the suspicious file that has appeared on the *Desktop.* kernel audit logs can be a great source to validate whether an entity has affected another entity or whether there is any correlation among some entities.

In this dissertation, we took the first step towards utilizing kernel audit logs for designing efficient threat detection systems. Nevertheless, this research area still has many challenges and open problems to solve. One major challenges would be to scale the proposed methods to an enterprise with thousands of machines where bandwidth, storage and processing limitations abound. In this situation, the cyber defenders often end up with a partially collected provenance graph in which it might not be possible to find the connectivity between entities by performing backward/forward traversals. As a future work, we can incorporate some characteristics of the entities (such as similarity between their names) in addition to backward/forward tracking results to make the correlation on a partially collected provenance graph.

APPENDICES

Appendix A

Sleuth's ADDITIONAL ATTACKS DISCUSSION

In this section, we provide graphs that reconstruct attack campaigns that weren't discussed in Section 3.6.4. Specifically, we discuss attacks L-1, F-1, F-2, W-1, and L-3.

Attack L-1. In this attack (Figure 25), `firefox` is exploited to drop and execute via a shell the file `mozillanightly`. The process `mozillanightly` first downloads and executes `mozillaautoup`, then starts a shell, which spawns several other processes. Next, the information gathered in file `netrecon.log` is exfiltrated and the file removed.

Attack F-1. In this attack (Figure 26), the `nginx` server is exploited to drop and execute via shell the file `dropper`. Upon execution, the `dropper` process forks a shell that spawns several processes, which write to a file and reads and writes to sensitive files. In addition, `dropper` communicates with the IP of the attacker. We report in the figure the graph related to the restoration and administration carried out after the engagement, as discussed in Section 3.6.5.

Attack F-2. The start of this attack (Figure Figure 27) is similar to F-1. However, upon execution, the `dropper` process downloads three files named `recon, sysman`, and `mailman`. Later, these files are executed and used which are used to exfiltrate data gathered from the system.

Attack W-1. In this attack (Figure 28), `firefox` is exploited twice to drop and execute a file `mozillanightly`. The first `mozillanightly` process downloads and executes the file `photosnap.exe`, which takes a screenshot of the victim's screen and saves it to a png file.

Appendix A (Continued)

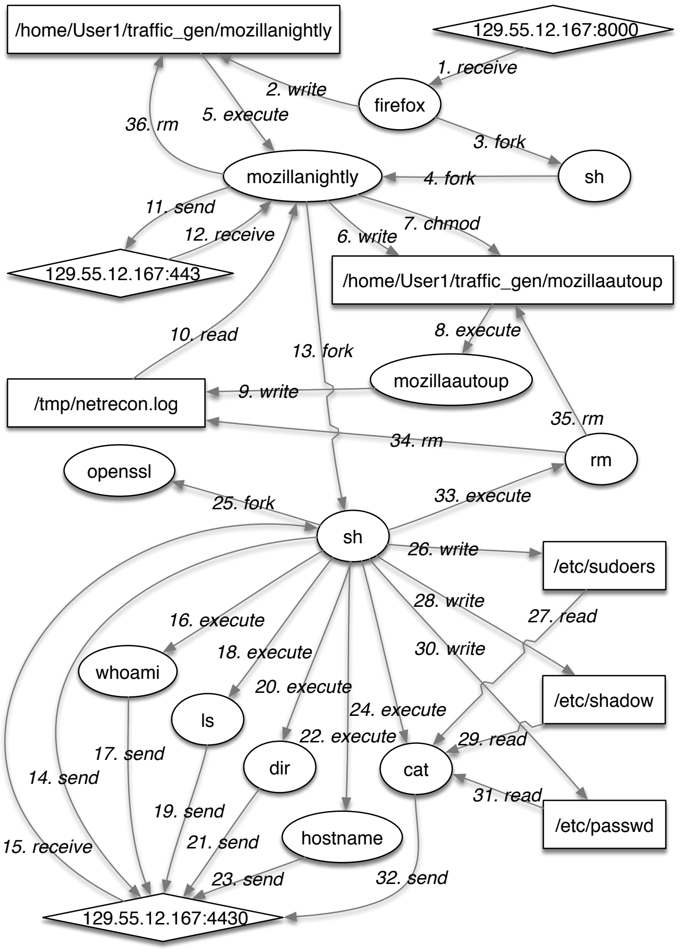


Figure 25. Scenario graph reconstructed from campaign L-1.

Subsequently, the jpeg file is exfiltrated by `mozillanightly`. The second `mozillanightly` process downloads and executes two files: 1) `burnout.bat`, which is read, and later used to issue commands to `cmd.exe` to gather data about the system; 2) `mnsend.exe`, which is executed by `cmd.exe` to exfiltrate the data gathered previously.

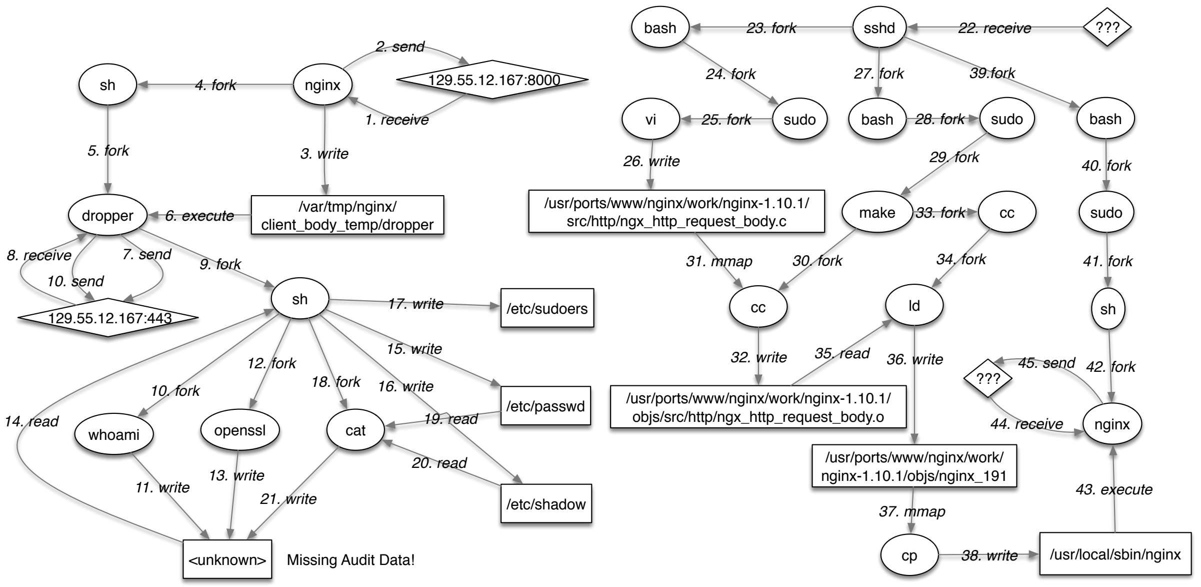


Figure 26. Scenario graph reconstructed from campaign F-1.

Attack L-3. In this attack (Figure 29), the file `dropbearLINUX.tar` is downloaded and extracted. Next, the program `dropbearkey` is executed to create three keys, which are read by a program `dropbear`, which subsequently performs exfiltration.

Appendix A (Continued)

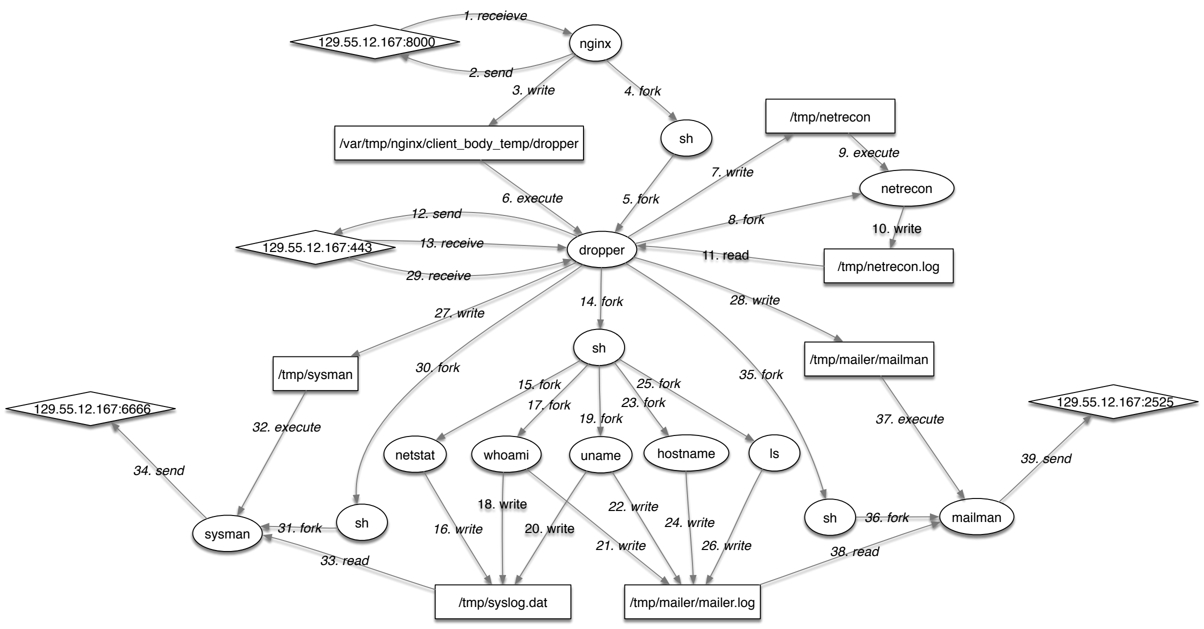


Figure 27. Scenario graph reconstructed from campaign F-2.

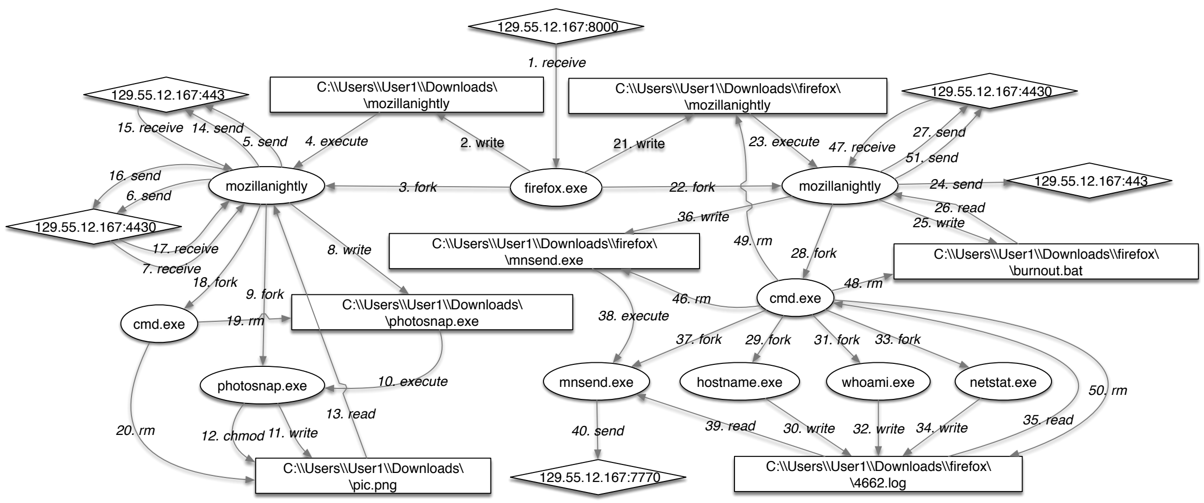


Figure 28. Scenario graph reconstructed from campaign W-1.

Appendix A (Continued)

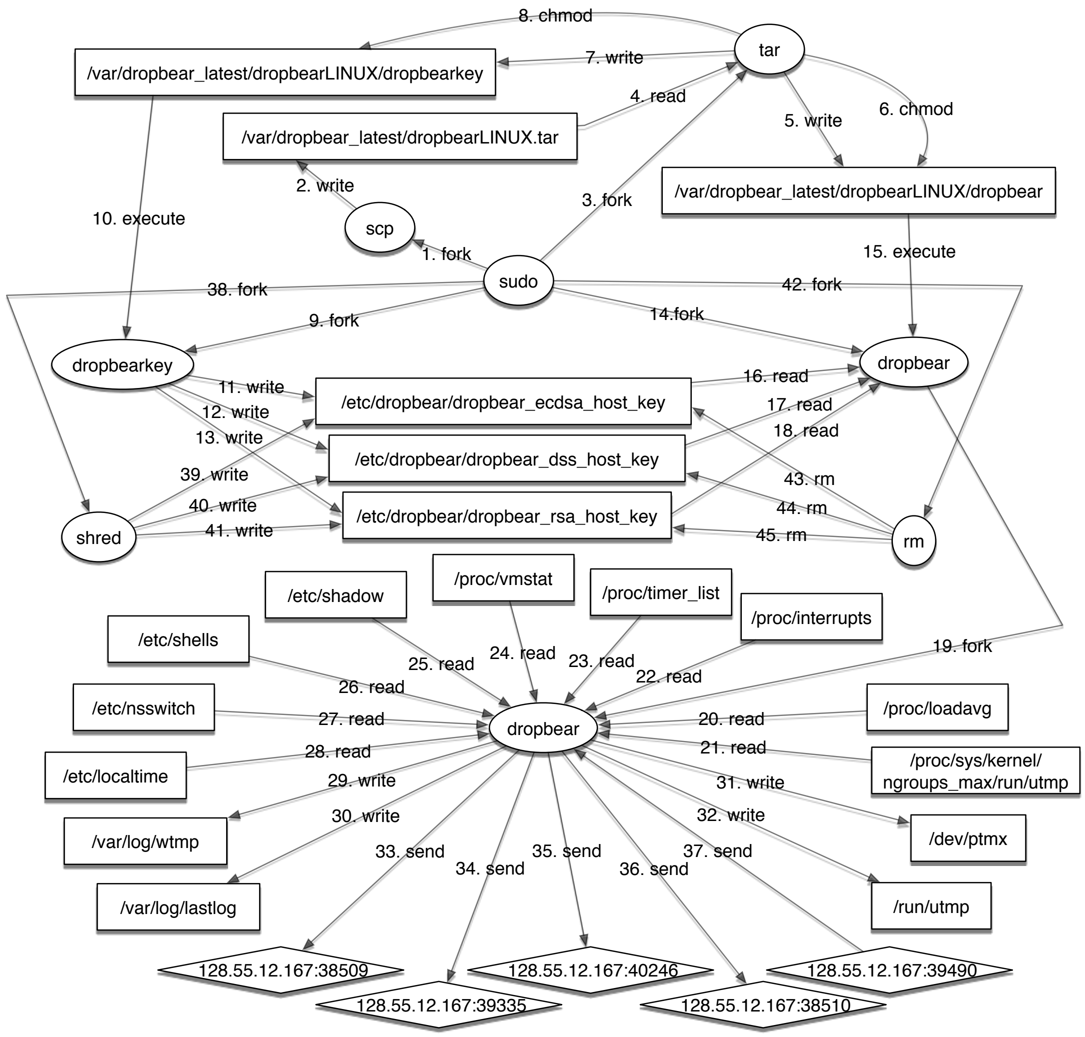


Figure 29. Scenario graph reconstructed from campaign L-3.

Appendix B

Holmes ATTACK SCENARIOS

Scenario-2: Trojan. This attack scenario (Fig. Figure 30) begins with a user downloading a malicious file. The user then executes the file. The execution results in a C&C communication channel with the attacker's machine. The attacker then launches a shell and executes some information gathering commands such as *hostname*, *whoami*, *ifconfig*, *netstat*, and *uname*. Finally, the attacker exfiltrates some secret files. Note that this attack scenario is similar to the *Drive-by Download* scenario discussed earlier except that the initial compromise happens via a program that the user downloads. Another important insight from the detection results of this scenario is that it was missing important events that are relevant to the C&C communication (*connect*) and final cleanup (*unlink*) activity of the attack. Even with such incomplete data, Holmes was able to flag this as an APT since the Threat score surpassed the threshold.

Scenario-3: Trojan. In this attack (Fig. Figure 31), a user is convinced to download a malicious Trojan program (texteditor) via Firefox. Next, the user moves the executable file to another directory, changes its name (tedit), and finally executes it. After the execution, a C&C channel is created, and a reverse shell is provided to the attacker. The attacker launches a shell prompt and executes information gathering commands like *hostname*, *whoami*, *ifconfig*, and *netstat*. The attacker then deploys another malicious file, exfiltrates information, and finally cleans up his footprints. This scenario differs from *Trojan-1* because it has an additional activity that remotely deploys a new malicious executable.

Appendix B (Continued)

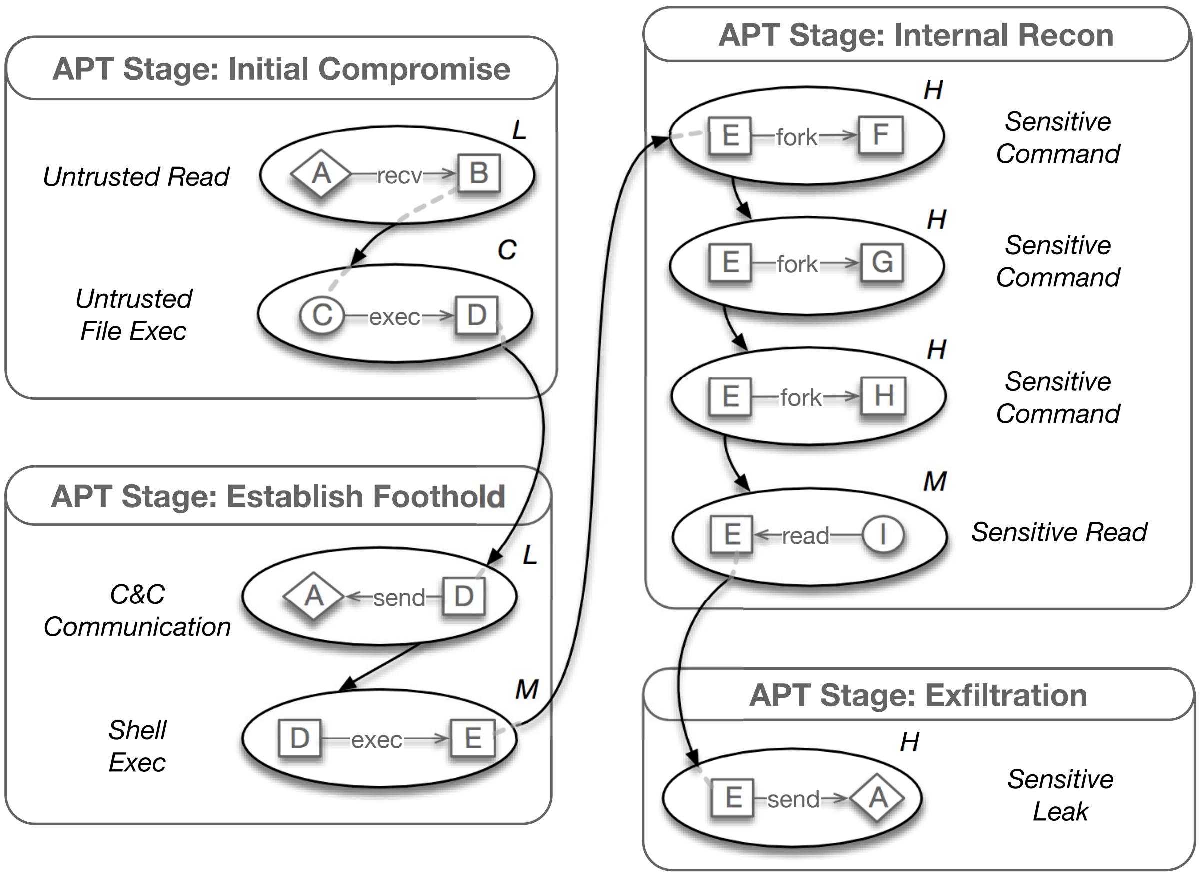


Figure 30. HSG of Scenario-2. Notations: A= Untrusted External Address; B= Firefox; C= Trojan File (diff); D= Executed Trojan Process; E= /bin/dash; F= ifconfig; G= hostname; H= netstat; I= password.txt;

Scenario-4: Spyware. This attack (Fig. Figure 32) begins when the red-team compromises Firefox. The user on the victim host then loaded a hijacked remote URL. Next, a shellcode from the URL is executed to connect to a C&C server from which it downloaded a malicious binary, wrote it to disk, and executed it. The execution of the malicious binary results in a reverse shell channel for C&C communications. The attacker then ran the shell command, resulting in a new *cmd.exe* process and a new connection to the C&C server. The operator ran reconnaissance commands (*hostname*, *whoami*, *ipconfig*, *netstat*, *uname*). The attacker then exfiltrated the *password.txt* file and then deleted it. Finally, the malicious binary drops a batch file that deletes attack footprints, including the malicious binary itself.

Scenario-5.1: Eternal Blue. This APT exploits vulnerable SMB [137] services in Windows. In this scenario (see Fig. Figure 33), Meterpreter [138] was used with the recently implemented Eternal Blue exploit and Double Pulsar reflective loading capabilities. The attacker exploited the listening SMB service on port 445 of the target. A shellcode was then downloaded and executed on the target. The shellcode performed process injection into the *lsass.exe* process. *lsass.exe* then launched *rundll32.exe*, which connected to the C&C server and downloaded-and-executed Meterpreter. Next, Meterpreter exfiltrated a sensitive file and cleared Windows event logs.

Scenario-5.2: RAT. In this attack (Fig. Figure 34), Firefox navigates to a malicious website and gets exploited. Then, a Remote Access Trojan (RAT) is uploaded to the victim's machine and executed. After execution, a connection to the C&C server has happened, and the malicious RAT is deleted. This attack scenario is incomplete, and no harm is done.

Scenario-6: Web-Shell. The assumption in this attack (Fig. Figure 35) is that *Nginx* web server has a vulnerability that gives the attacker access to run arbitrary commands on the server (similar to Shellshock bug). As a result, the attacker exfiltrates a sensitive file. The important insight here is that by capturing sufficiently strong APT signals of an ongoing attack through TTP matching, HOLMES accurately flags an APT, even when a critical APT step is missing (initial compromise in this case).

Scenario-7.1: RAT. A vulnerable *Nginx* server was installed during the setup period. The attacker exploits the *Nginx* server by throwing a malicious shell-code. *Nginx* runs the malicious shell-code which results in the download and execution of a malicious RAT. Next, RAT connects to a C&C server and gives administrative privileges to the remote attacker. The attacker remotely executes some commands. It then deploys some malicious Python scripts and exfiltrates information. The HSG of this attack is shown in Fig. Figure 36.

Appendix B (Continued)

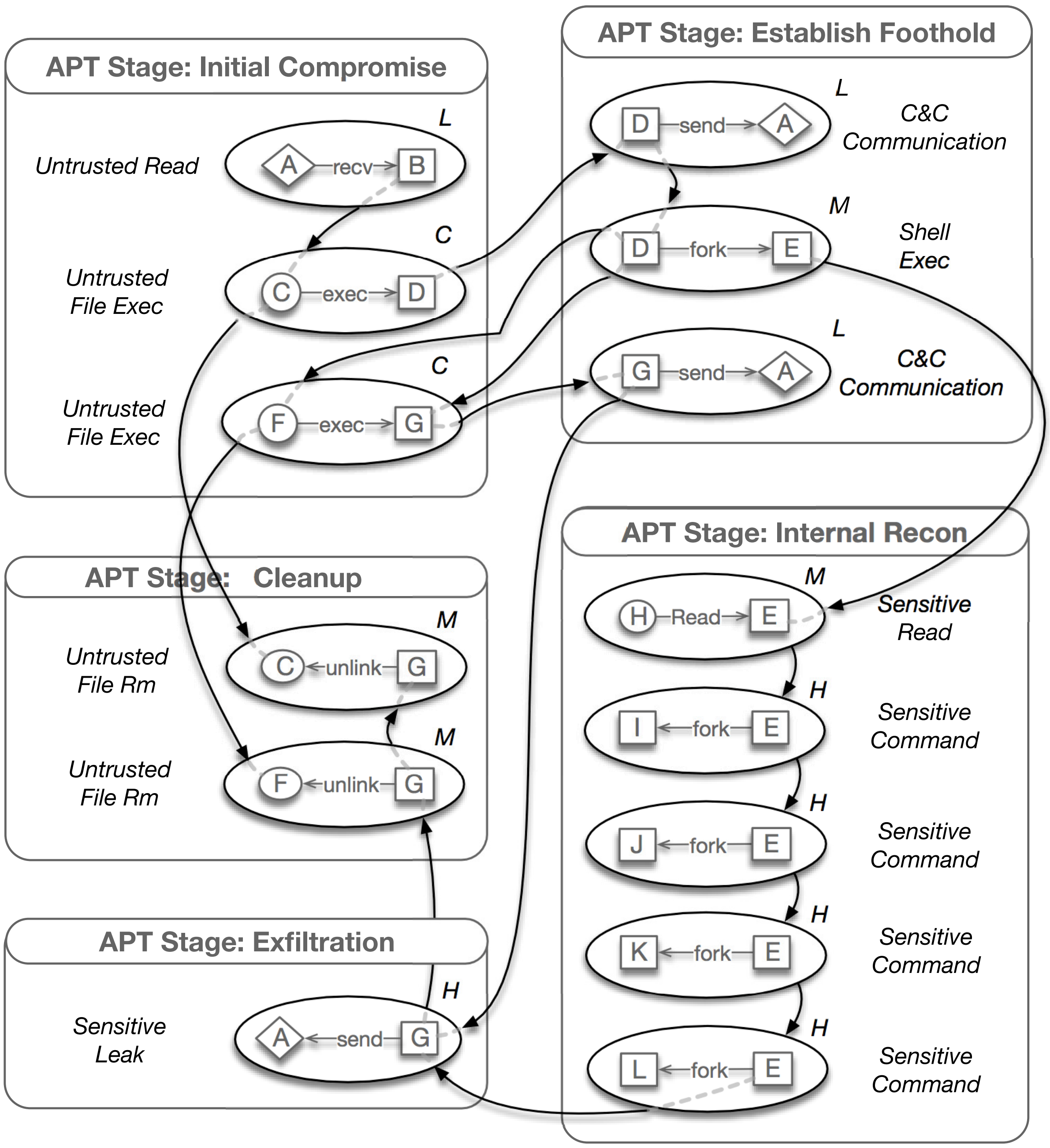


Figure 31. HSG of Scenario-3. Notations: A= Untrusted External Address; B= Firefox; C= Trojan File (tedit); D= Executed Trojan Process; E= /bin/dash; F= Malicious Executable file (py); G= Executed Malicious Process; H= password.txt; I= whoami; J= ifconfig; K= netstat; L= uname;

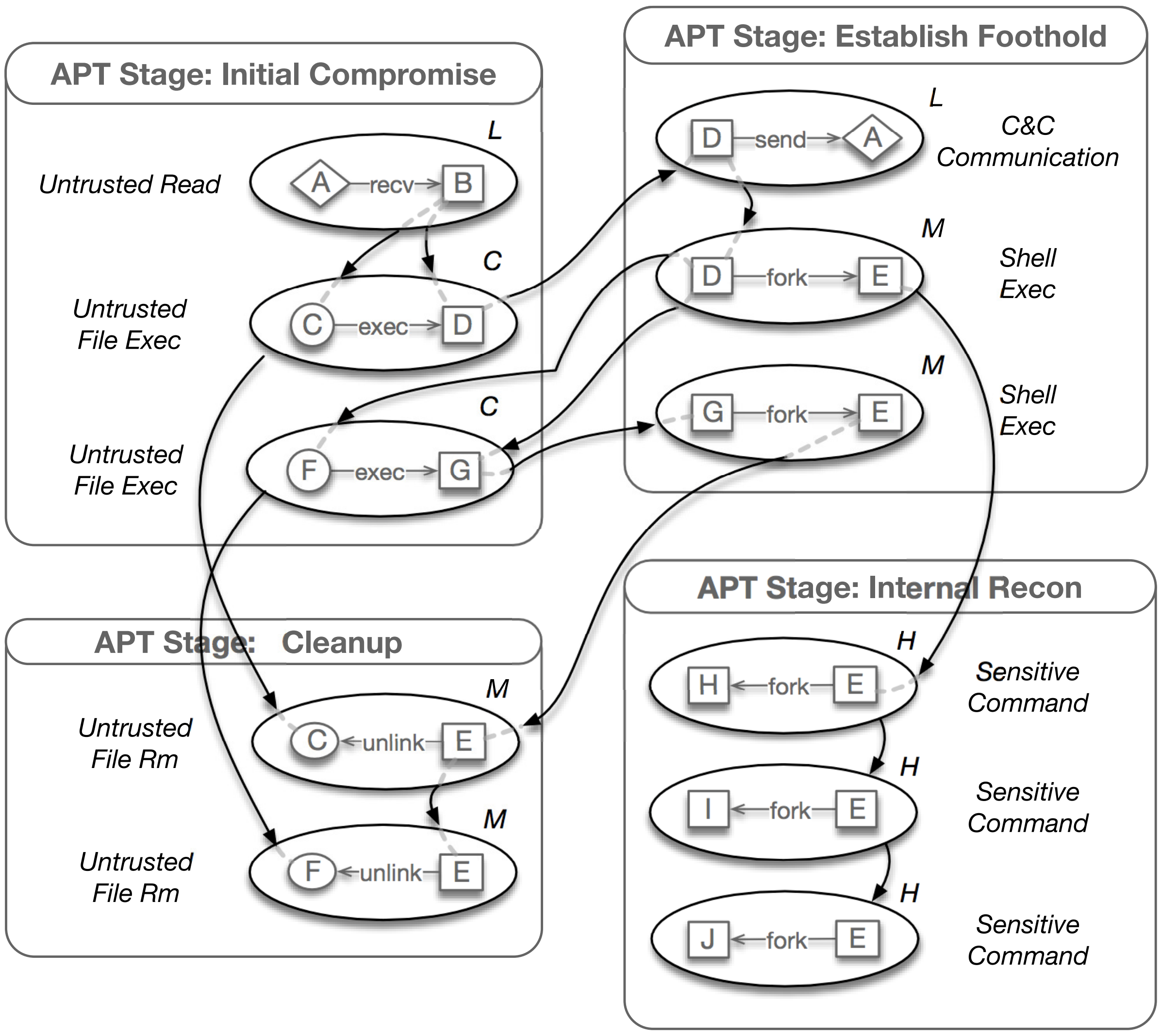


Figure 32. HSG of Scenario-4. Notations: A= Untrusted External Address; B= Firefox.exe; C= Malicious dropped file (procman.exe); D= Executed Malware Process; E= cmd.exe; F= Malicious Batch file (burnout.bat); G= Executed Batch Process; H= hostname; I= whoami; J= ipconfig;

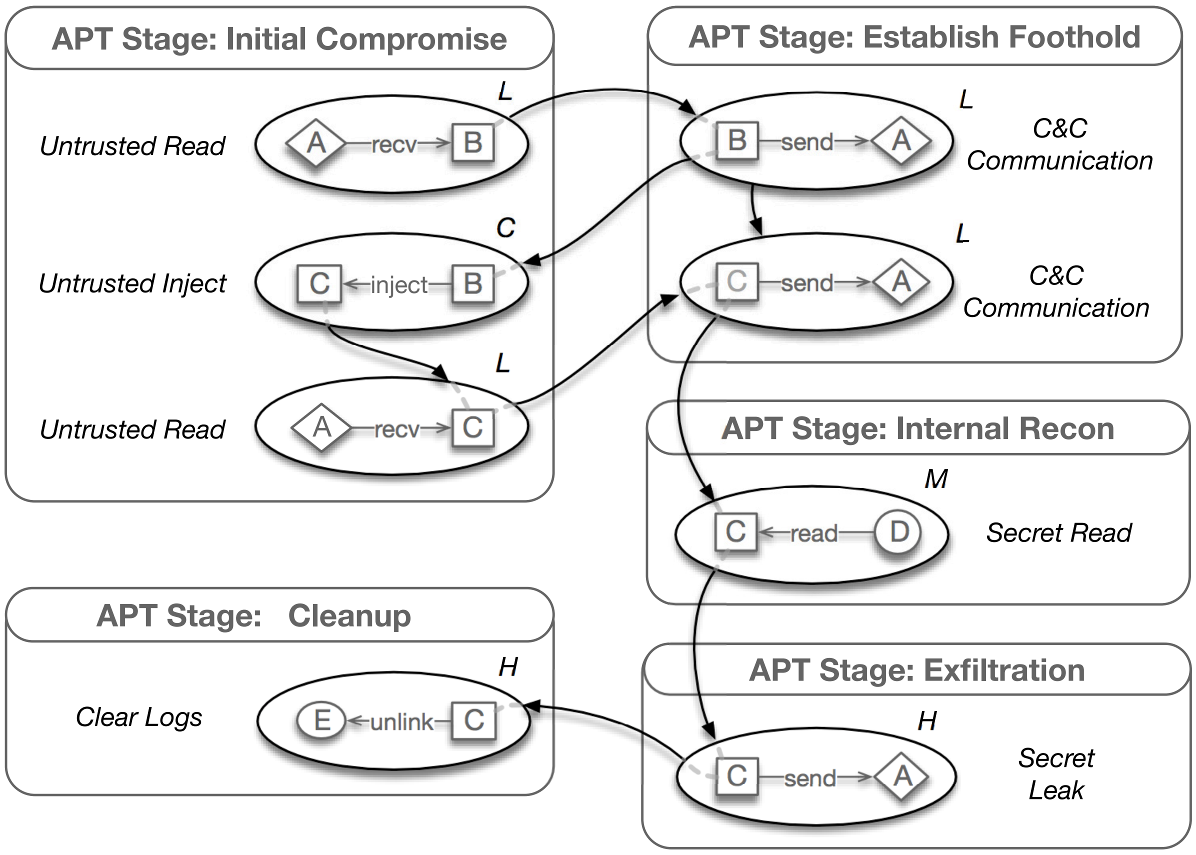


Figure 33. HSG of Scenario-5.1 (Eternal Blue). Notations: A= Untrusted External Address; B= lsass.exe; C= rundll32.exe; D= password.txt; E= Winevt logs;

Appendix B (Continued)

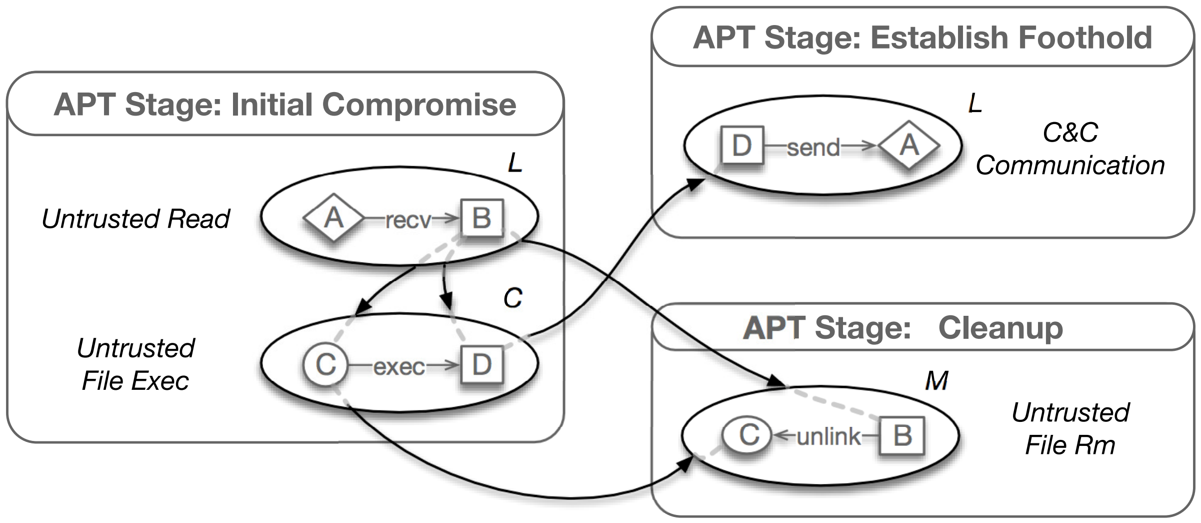


Figure 34. HSG of Scenario-5.2. Notations: A= Untrusted External Address; B= Firefox.exe; C= Malicious dropped file (spd.exe); D= Executed Malware Process;

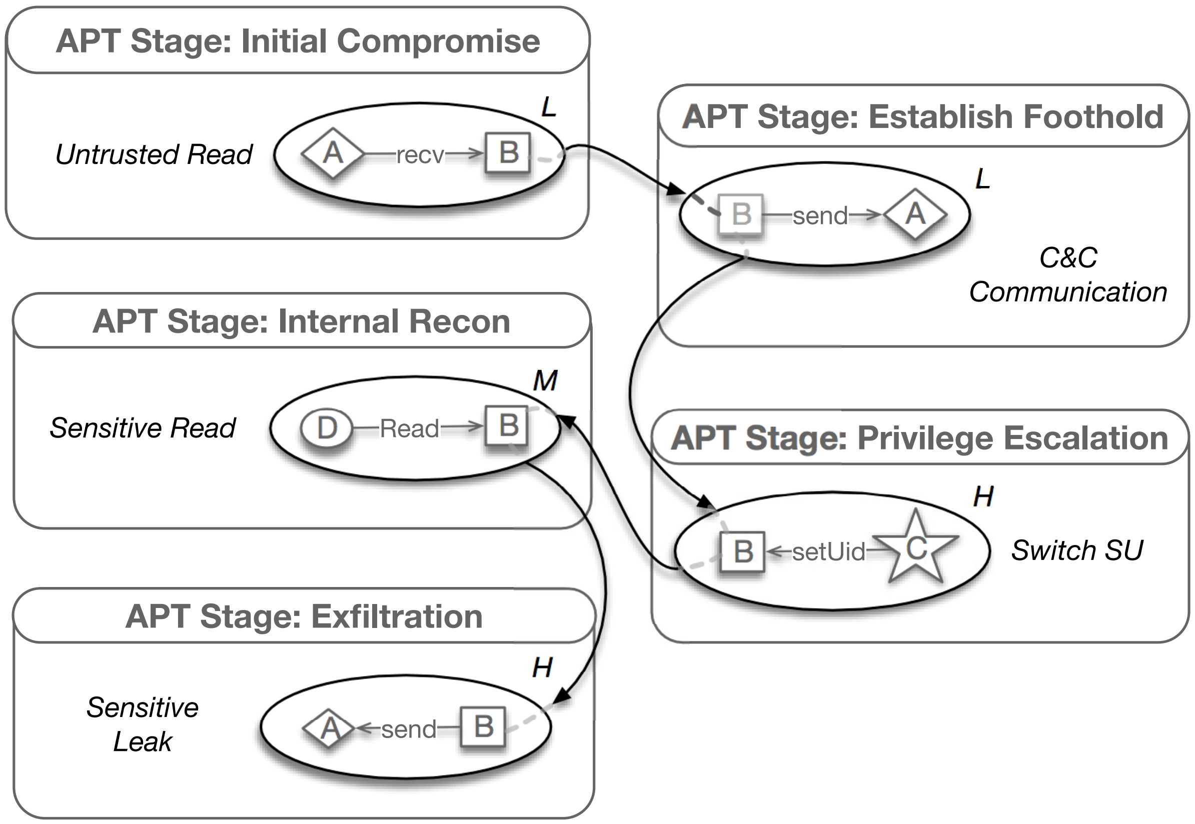


Figure 35. HSG of Scenario-6. Notations: A= Untrusted External Address; B= Nginx; C= Root userID; D= Passwd.txt;

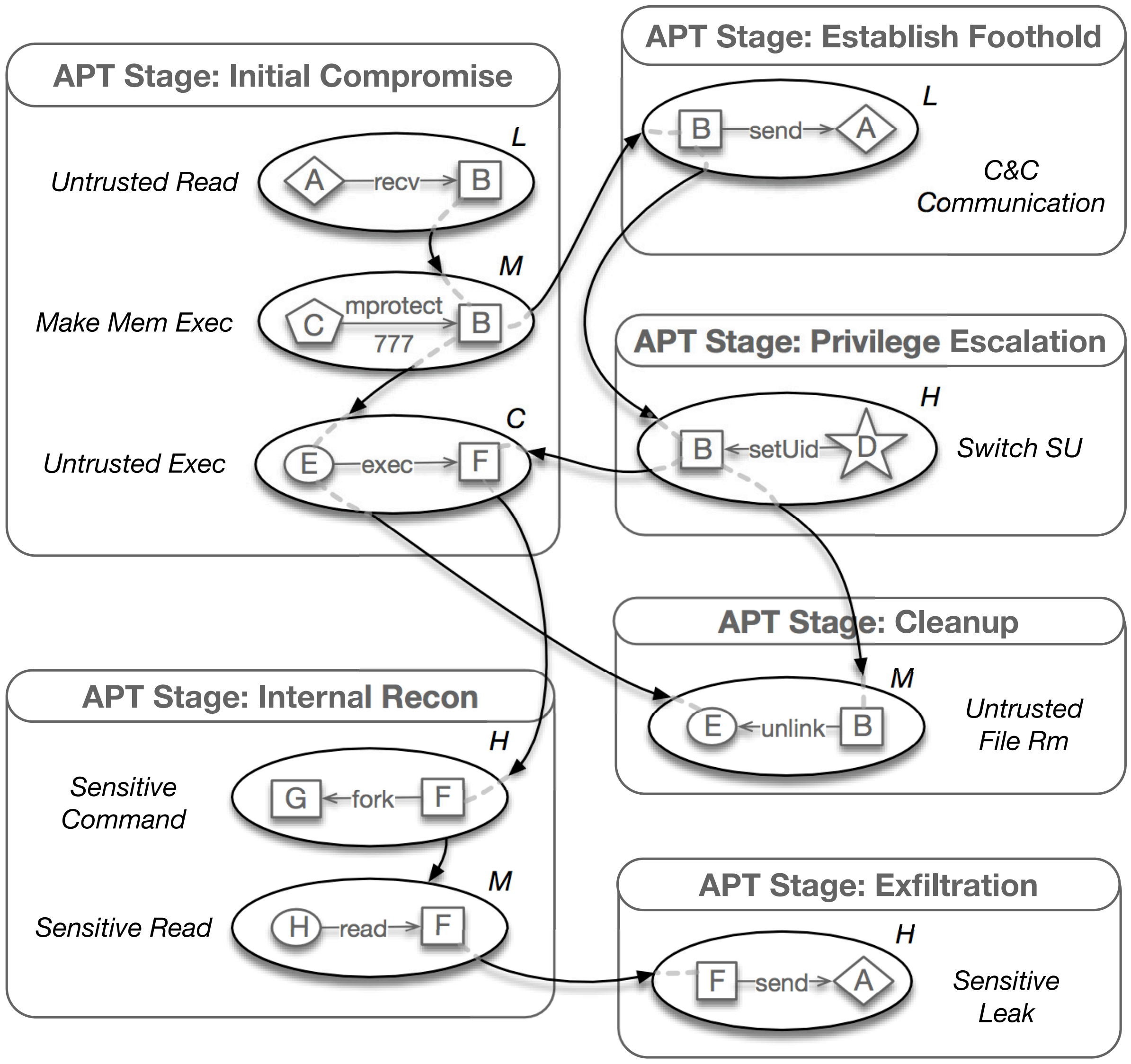


Figure 36. HSG of Scenario-7.1. Notations: A= Untrusted External Address; B= Nginx; C= Memory; D= Root userID; E= Malicious dropped file (py); F= Executed Malware Process; G= uname; H= /etc/shadow;

Appendix C

Poirot CTI DESCRIPTIONS

In this section, we provide a brief history of each malware and a summary of the statements from their corresponding reports which we have used to construct the query graphs.

njRAT. njRAT is a publicly available Remote Access Trojan (RAT) that gives the attacker full control over the victim system. Although the source code of njRAT is publicly available, attacks leveraging njRAT have mostly targeted organizations based in or focused on the Middle East region in the government, telecom, and energy sectors. When the malware is executed, it tries to read its configuration from a file with the extension of ".exe.config" (edge 1). njRAT malware stores the logged keystrokes in a ".tmp" file (edge 2), and also writes to a ".pf" file (edge 3). To gain persistence, njRAT malware creates some copies of itself (edges 4&8). After execution (edges 5&6), one of the copies writes to a ".pf" file (edge 7). njRAT malware also start a netsh process located at (edge 9), which results in creation of another ".pf" file (edge 10). Finally, the malware sets some registry values (edges 11-13) and beacons to a C&C server at 217.66.231.245 (edge 14).

DeputyDog. DeputyDog refers to a malware appearing to have targeted organizations in Japan, based on a report by FireEye. The query graph that we extracted from the report of this malware is shown in Figure 19, and it is described in 5.2.

Uroburos. Uroburos, ComRAT, Snake, Turla, and Agent.BTZ are all referring to a family of rootkit which is responsible for the most significant breach of U.S. military computers. The malware starts by dropping two Microsoft Windows dynamic libraries (edges 1&2) and calling rundll32.exe (edge 3) to install these libraries (edges 4&5). Then, to be started during the boot process, the malware creates a registry key (edge 6). The malware creates three log files (edges 7-9) and removes a set of file (edges 10-14).

Carbanak. Carbanak is a remote backdoor to provide remote access to infected machines. The main motivation of the attackers appears to be financial gain, which has resulted in cumulative losses up to one billion dollars [75]. The compromise initially starts using a spear phishing email that appears to be legitimate banking communications (edge 1). After the exploit, Carbanak copies itself into "%system32%" with the name "svchost.exe" (edges 2-4) and deletes the original file created by the exploit payload (edge 5). To access autorun privileges, the malware creates a new service with a name in the format of "<ServiceName>Sys", where ServiceName is any existing service randomly chosen (edge 6). Carbanak creates a file with a random name and a .bin extension where it stores commands to be executed (edge 7). Then, the malware gets the proxy configuration from a registry entry (edge 8) and the Mozilla Firefox configuration file (edge 9). Finally, Carbanak communicates with its C&C server (edge 10).

DustySky. DustySky is a multi-stage malware whose main objective is intelligence gathering for political purposes. The malware sample is disguised as a Microsoft Word file, and once it is executed (edge 1), a lure Microsoft word document in the Arabic language is opened (edges 2&3) while the malware performs intelligence gathering in the background. For VM evasion, the dropper checks the existence of some DLL files, specifically vboxmrxnp.dll and vmbusres.dll which indicate existence of VirtualBox (edges 4&5) and vmGuestlib.dll which indicates existence of VMware (edge 6). DustySky Core is dropped to %TEMP% (edges 7&8&9), and keystroke logs are saved to %TEMP%\temps (edge 10).

OceanLotus. OceanLotus, also known as APT32, is believed to be a Vietnam-based APT group targeting Southeast Asian countries. After execution of this malware (edge 1), a decoy document and an eraser application are dropped (edges 2&3), and the decoy document is lunched in Microsoft Word (edges 4&5). Then, the executable decrypts its resources and drops a copy of legitimate Symantec Network Access Control application (edge 6), an encrypted backdoor (edge 7), and a malicious DLL file (edge 8). The Symantec application, which is signed and legitimate, loads all the libraries in the same folder by default. In this case, after execution (edges 9&10), this application loads the malicious DLL file which has been dropped in the same directory (edge 11). It then reads the backdoor file (edge 12) which results in accessing a registry (edge 13), loading the HTTPProv.dll library (edge 14), and creating a registry key (edge 15). Finally, the malware connects to its mothership (edges 16&17).

HawkEye. HawkEye is a malware-as-a-service credential stealing malware and is a popular tool among APT campaigns. The new variant of this malware uses process hollowing to inject its code into the legitimate signed .NET framework executables and ships with many sophisticated functions to evade detection. This new variant is usually delivered as a compressed file, and after decompression (edges 1&2) and execution(edge 3), it spawns a child process (edge 4), called RegAsm, which is an assembly registration tool from the Microsoft .Net framework. HawkEye extracts a PE file into its memory and then injects it into the RegAsm process. After sleeping for 10 seconds, the RegAsm process spawns two child processes named vbc both from the .Net framework as well (edges 5&6). One of these processes collects credentials of browsers, while the other one focuses on email and Instant Messaging (IM) appllications. We have added one node, typed as a file or registry, corresponding to the name of each browser (edges 7-18) or email/IM (edges 19-26) application mentioned in the report. Note that these applications might store some confidential information of interest to attackers into both files or registries, and that is why we did not limit our search to only files or registries. The collected credentials are regularly saved into ∗.tmp files in the %temp% directory (edges 27&28), while after a while, the RegAsm process reads the entire data of these tmp files into its memory (edges 29&30) and deletes them immediately (edges 31&32). Finally, RegAsm looks up the machines public IP from “http[s]:\\whatismyipaddress.com\” web service (edges 33&34) and then exfiltrates the collected information to the attacker’s email address (edge 35).

It is important to note that there are some nodes with exactly same label and type in the query graph of HawkEye, such as F&G or J&K. However, these nodes get aligned to different nodes based on their dependencies with other entities. For example, node F interacts with browser applications while node G interacts with the email/IM applications. In addition, the alignment of browser or mail application nodes is independent of their installation on the system. Many of these applications are not installed on the test machine, however when the malware attempts to check whether these applications are installed on the system, it initiates an OPEN event which gets detected by Poirot.

Appendix D

COPYRIGHT PERMISSION STATEMENT

DocuSign Envelope ID: 0BE9773A-0101-4F69-B24E-04E6C1D9E1C8

usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

CONSENT FORM for REFEREED PAPERS
USENIX Security '17: 26th USENIX Security Symposium
August 16–18, 2017, Vancouver, BC, Canada

AUTHORIZATION AND LICENSE TO PUBLISH

A major mission of the USENIX Association ("USENIX") is to provide for the creation and dissemination of new knowledge. In order to facilitate this process, USENIX allows authors to retain ownership of the copyright in their works, requesting only that USENIX be granted the right to publish that work. Therefore, the parties agree as follows:

1. **Grant of Rights to Publish.** You hereby grant to The USENIX Association ("USENIX"), the right to publish and re-publish the paper provisionally entitled:

Paper Title: SLEUTH: Real-time Attack Scenario Reconstruction from COTS Audit Data

PLEASE FILL IN THE TITLE HERE

(the "Paper") in printed form, electronically, and in audiovisual formats (the "Rights"). USENIX's rights are non-exclusive, worldwide, perpetual and irrevocable, but also non-sublicensable and non-transferable. Among the rights you hereby grant to USENIX is the right to be the first publisher of the Paper. USENIX initially plans to publish the Article in the following publication: **Proceedings of the 26th USENIX Security Symposium (USENIX Security '17)** (the "Original Publication") and to mount the files permanently for display on the USENIX Web site, but may also otherwise disseminate the Paper, as well as the oral presentation of the Paper, on media, as a live broadcast, and in any other electronic or audiovisual format chosen by USENIX.

2. **Posting.** You retain all rights in the Paper subject to the non-exclusive rights granted to USENIX in Section 1 above.

3. **Notice Request.** USENIX will include the following notice on the first page of the Original Publication: "Copyright to this work is retained by the author[s]."

4. **Subsequent Publication by Author.** You must include a notice of original publication by USENIX in any subsequent publication of the Paper or a revised version of the Paper.

5. **Warranties.** You warrant that:
   a. You are the sole author and owner of the Paper and the rights pertaining thereto or, if the Paper is a work of joint authorship, that all of the owners of the Paper have signed and submitted a copy of this document. If the Paper is owned by your employer, you warrant that you have obtained permission to enter into this agreement and to grant the rights herein described to USENIX.
   b. USENIX does not need the permission of any other party to exercise the rights granted herein.
   c. The Paper does not contain any defamatory or other unlawful content.
   d. The Paper does not infringe upon the rights of others.
   e. The Paper and its contents are original to you.
   f. The Paper is not currently submitted for publication elsewhere and has not been previously published elsewhere.
   g. If the Paper contains significant excerpts from other copyrighted materials, written permission from all of the copyright owners of those materials have been obtained, copies of such permissions are attached to this document, and proper credit and attribution has been given in the Paper itself.

6. **Right to Withdraw Access.** USENIX reserves the right to remove content and publications under its control from public access at any time for any reason. Occasionally, USENIX receives notices alleging that the content on its Web site, in its own publications, or in the papers and conference proceedings it publishes may infringe third party rights or may otherwise violate some law or regulation. USENIX does not usually take a position on the merits of these allegations. However, when this happens, USENIX will inform the authors of the allegations and may remove the identified content or publication from public access. USENIX may restore the material, at its discretion, when it deems appropriate. USENIX may also send the notice containing the allegations to Lumen (lumendatabase.org).

USENIX Code of Conduct Guidelines

Speakers are responsible for the content of their presentations, but USENIX requests that speakers be cognizant of potentially offensive actions, language, or imagery, and that they consider whether it is necessary to convey their message. If they do decide to include it, USENIX asks that they warn the audience, at the beginning of the talk, and provide them with with the opportunity to leave the room to avoid seeing or hearing the material.

Your signature below grants USENIX the right to publish and also indicates that you have read the USENIX Event Code of Conduct and Guidelines for Speakers available at www.usenix.org/conferences/coc.

Date: 6/28/2017 | 09:09 PDT

Signature of Author(s) DocuSigned by: Md Nahid Hossain 7D30700D66EC434...

Printed Name(s) of Author(s) Md Nahid Hossain

Authorized Signature for USENIX Michele Nelson

Certificate Of Completion

Envelope Id: 0BE9773A01014F69B24E04E6C1D9E1C8		Status: Completed
Subject: Please DocuSign: USENIX Security '17 Consent Form for Refereed Papers		
Source Envelope:		
Document Pages: 1	Signatures: 1	Envelope Originator:
Supplemental Document Pages: 0	Initials: 0	Consent Forms
Certificate Pages: 1		
AutoNav: Enabled	Payments: 0	2560 9th St, Ste 215
EnvelopeId Stamping: Enabled		Berkeley, CA 94710
Time Zone: (UTC-08:00) Pacific Time (US & Canada)		consent@usenix.org
		IP Address: 130.245.137.80

Record Tracking

Status: Original 6/28/2017 9:05:26 AM	Holder: Consent Forms consent@usenix.org	Location: DocuSign

Signer Events	Signature	Timestamp
Md Nahid Hossain mdnhossain@cs.stonybrook.edu Security Level: Email, Account Authentication (None)	DocuSigned by: Md Nahid Hossain 7D30700D66EC434... Using IP Address: 130.245.137.80	Sent: 6/28/2017 9:05:26 AM Viewed: 6/28/2017 9:06:11 AM Signed: 6/28/2017 9:09:20 AM

Electronic Record and Signature Disclosure:
Not Offered via DocuSign

In Person Signer Events	Signature	Timestamp

Editor Delivery Events	Status	Timestamp

Agent Delivery Events	Status	Timestamp

Intermediary Delivery Events	Status	Timestamp

Certified Delivery Events	Status	Timestamp

Carbon Copy Events	Status	Timestamp

Notary Events	Signature	Timestamp

Envelope Summary Events	Status	Timestamps
Envelope Sent	Hashed/Encrypted	6/28/2017 9:05:26 AM
Certified Delivered	Security Checked	6/28/2017 9:06:12 AM
Signing Complete	Security Checked	6/28/2017 9:09:20 AM
Completed	Security Checked	6/28/2017 9:09:20 AM

Payment Events	Status	Timestamps

Consent to Publish
Lecture Notes in Computer Science

Title of the Book or Conference Name: 14th International Conference on Information Systems S

Volume Editor(s) Name(s): R.K. Shyamasundar, Vinod Ganapathy, and Trent Jaeger

Title of the Contribution: ProPatrol: Attack Investigation via Extracted High-Level Tasks

Author(s) Full Name(s): Sadegh Momeni Milajerdi, Birhanu Eshete, Rigel Gjomemo, Venkat V

Corresponding Author's Name, Affiliation Address, and Email:
Sadegh Momeni Milajerdi, "University of Illinois at Chicago, 1200 W Harrison St, Chicago, IL 60607, USA", smomen2@uic.edu

When Author is more than one person the expression "Author" as used in this agreement will apply collectively unless otherwise indicated.

The Publisher intends to publish the Work under the imprint **Springer**. The Work may be published in the book series **Lecture Notes in Computer Science (LNCS, LNAI or LNBI)**.

§ 1 Rights Granted

Author hereby grants and assigns to **Springer Nature Switzerland AG, Gewerbestrasse 11, 6330 Cham, Switzerland** (hereinafter called **Publisher**) the exclusive, sole, permanent, world-wide, transferable, sub-licensable and unlimited right to reproduce, publish, distribute, transmit, make available or otherwise communicate to the public, translate, publicly perform, archive, store, lease or lend and sell the Contribution or parts thereof individually or together with other works in any language, in all revisions and versions (including soft cover, book club and collected editions, anthologies, advance printing, reprints or print to order, microfilm editions, audiograms and videograms), in all forms and media of expression including in electronic form (including offline and online use, push or pull technologies, use in databases and data networks (e.g. the Internet) for display, print and storing on any and all stationary or portable end-user devices, e.g. text readers, audio, video or interactive devices, and for use in multimedia or interactive versions as well as for the display or transmission of the Contribution or parts thereof in data networks or search engines, and posting the Contribution on social media accounts closely related to the Work), in whole, in part or in abridged form, in each case as now known or developed in the future, including the right to grant further time-limited or permanent rights. Publisher especially has the right to permit others to use individual illustrations, tables or text quotations and may use the Contribution for advertising purposes. For the purposes of use in electronic forms, Publisher may adjust the Contribution to the respective form of use and include links (e.g. frames or inline-links) or otherwise combine it with other works and/or remove links or combinations with other works provided in the Contribution. For the avoidance of doubt, all provisions of this contract apply regardless of whether the Contribution and/or the Work itself constitutes a database under applicable copyright laws or not.
The copyright in the Contribution shall be vested in the name of Publisher. Author has asserted his/her right(s) to be identified as the originator of this Contribution in all editions and versions of the Work and parts thereof, published in all forms and media. Publisher may take, either in its own name or in that of Author, any necessary steps to protect the rights granted under this Agreement against infringement by third parties. It will have a copyright notice inserted into all editions of the Work according to the provisions of the Universal Copyright Convention (UCC).
The parties acknowledge that there may be no basis for claim of copyright in the United States to a Contribution prepared by an officer or employee of the United States government as part of that person's official duties. If the Contribution was performed under a United States government contract, but Author is not a United States government employee, Publisher grants the United States government royalty-free permission to reproduce all or part of the Contribution and to authorise others to do so for United States government purposes. If the Contribution was prepared or published by or under the direction or control of the Crown (i.e., the constitutional monarch of the Commonwealth realm) or any Crown government department, the copyright in the Contribution shall, subject to any agreement with Author, belong to the Crown. If Author is an officer or employee of the United States government or of the Crown, reference will be made to this status on the signature page.

§ 2 Rights Retained by Author

Author retains, in addition to uses permitted by law, the right to communicate the content of the Contribution to other research colleagues, to share the Contribution with them in manuscript form, to perform or present the Contribution or to use the content for non-commercial internal and educational purposes, provided the original source of publication is cited according to the current citation standards in any printed or electronic materials. Author retains the right to republish the Contribution in any collection consisting solely of Author's own works without charge, subject to ensuring that the publication of the Publisher is properly credited and that the relevant copyright notice is repeated verbatim. Author may self-archive an author-created version of his/her Contribution on his/her own website and/or the repository of Author's department or faculty. Author may also deposit this version on his/her funder's or funder's designated repository at the funder's request or as a result of a legal obligation. He/she may not use the Publisher's PDF version, which is posted on the Publisher's platforms, for the purpose of self-archiving or deposit. Furthermore, Author may only post his/her own version, provided acknowledgment is given to the original source of publication and a link is inserted to the published article on the Publisher's website. The link must be provided by inserting the DOI number of the article in the following sentence: "The final authenticated version is available online at https://doi.org/[insert DOI]." The DOI (Digital Object Identifier) can be found at the bottom of the first page of the published paper.
Prior versions of the Contribution published on non-commercial pre-print servers like ArXiv/CoRR and HAL can remain on these servers and/or can be updated with Author's accepted version. The final published version (in pdf or html/xml format) cannot be used for this purpose. Acknowledgment needs to be given to the final publication and a link must be inserted to the published Contribution on the Publisher's website, by inserting the DOI number of the article in the following sentence: "The final authenticated publication is available online at https://doi.org/[insert DOI]".
Author retains the right to use his/her Contribution for his/her further scientific career by including the final published paper in his/her dissertation or doctoral thesis provided acknowledgment is given to the original source of publication. Author also retains the right to use, without having to pay a fee and without having to inform the Publisher, parts of the Contribution (e.g. illustrations) for inclusion in future work. Authors may publish an extended version of their proceedings paper as a journal article provided the following principles are adhered to: a) the extended version includes at least 30% new material, b) the original publication is cited, and c) it includes an explicit statement about the increment (e.g., new results, better description of materials, etc.).

§ 3 Warranties

Author agrees, at the request of Publisher, to execute all documents and do all things reasonably required by Publisher in order to confer to Publisher all rights intended to be granted under this Agreement. Author warrants that the Contribution is original except for such excerpts from copyrighted works (including illustrations, tables, animations and text quotations) as may be included with the permission of the copyright holder thereof, in which case(s) Author is required to obtain written permission to the extent necessary and to indicate the precise sources of the excerpts in the manuscript. Author is also requested to store the signed permission forms and to make them available to Publisher if required.
Author warrants that Author is entitled to grant the rights in accordance with Clause 1 "Rights Granted", that Author has not assigned such rights to third parties, that the Contribution has not heretofore been published in whole or in part, that the Contribution contains no libellous or defamatory statements and does not infringe on any copyright, trademark, patent, statutory right or proprietary right of others, including rights obtained through licences; and that Author will indemnify Publisher against any costs, expenses or damages for which Publisher may become liable as a result of any claim which, if true, would constitute a breach by Author of any of Author's representations or warranties in this Agreement.
Author agrees to amend the Contribution to remove any potential obscenity, defamation, libel, malicious falsehood or otherwise unlawful part(s) identified at any time. Any such removal or alteration shall not affect the warranty and indemnity given by Author in this Agreement.

§ 4 Delivery of Contribution and Publication

Author agrees to deliver to the responsible Volume Editor (for conferences, usually one of the Program Chairs), on a date to be agreed upon, the manuscript created according to the Publisher's Instructions for Authors. Publisher will undertake the reproduction and distribution of the Contribution at its own expense and risk. After submission of the Consent to Publish form signed by the Corresponding Author, changes of authorship, or in the order of the authors listed, will not be accepted by the Publisher.

§ 5 Author's Discount for Books

Author is entitled to purchase for his/her personal use (if ordered directly from Publisher) the Work or other books published by Publisher at a discount of 40% off the list price for as long as there is a contractual arrangement between Author and Publisher and subject to applicable book price regulation.
Resale of such copies is not permitted.

§ 6 Governing Law and Jurisdiction

If any difference shall arise between Author and Publisher concerning the meaning of this Agreement or the rights and liabilities of the parties, the parties shall engage in good faith discussions to attempt to seek a mutually satisfactory resolution of the dispute. This agreement shall be governed by, and shall be construed in accordance with, the laws of Switzerland. The courts of Zug, Switzerland shall have the exclusive jurisdiction.

Corresponding Author signs for and accepts responsibility for releasing this material on behalf of any and all Co-Authors.

Signature of Corresponding Author: **Date:**

[illegible] 10/08/2018

☐ I'm an employee of the US Government and transfer the rights to the extent transferable (Title 17 §105 U.S.C. applies)

☐ I'm an employee of the Crown and copyright on the Contribution belongs to the Crown

For internal use only:
Legal Entity Number: 1128 Springer Nature Switzerland AG
Springer-C-CTP-01/2018

IEEE COPYRIGHT LICENSE FORM FOR USE WITH IEEE COMPUTER SOCIETY'S EXPERIMENTAL "DELAYED-OPEN ACCESS" MODEL

TITLE OF WORK (hereinafter, "the Work"): HOLMES: Real-time APT Detection through Correlation of Suspicious Information Flows

AUTHOR(S): Sadegh Momeni Milajerdi, Rigel Gjomemo, Birhanu Eshete, R. Sekar, Venkat Venkatakrishnan

IEEE PUBLICATION TITLE: 2019 IEEE Symposium on Security and Privacy (SP)

Sadegh Momeni Milajerdi [illegible]
AUTHOR

09.25.2018
DATE

JOINT AUTHORSHIP

ACM Copyright and Audio/Video Release

Title of the Work: POIROT: Aligning Attack Behavior with Kernel Audit Records for Cyber Threat Hunting
Submission ID:fv607
Author/Presenter(s): Sadegh M. Milajerdi (UIC); Birhanu Eshete (Univ. of Michigan-Dearborn); Rigel Gjomemo (UIC); V.N. Venkatakrishnan (UIC);
Type of material: Full Paper

Publication and/or Conference Name: CCS '19: 2019 ACM SIGSAC Conference on Computer and Communications Security Proceedings

I. **Copyright Transfer, Reserved Rights and Permitted Uses**

* Your Copyright Transfer is conditional upon you agreeing to the terms set out below.

Copyright to the Work and to any supplemental files integral to the Work which are submitted with it for review and publication such as an extended proof, a PowerPoint outline, or appendices that may exceed a printed page limit, (including without limitation, the right to publish the Work in whole or in part in any and all forms of media, now or hereafter known) is hereby transferred to the ACM (for Government work, to the extent transferable) effective as of the date of this agreement, on the understanding that the Work has been accepted for publication by ACM.

Reserved Rights and Permitted Uses

(a) All rights and permissions the author has not granted to ACM are reserved to the Owner, including all other proprietary rights such as patent or trademark rights.

(b) Furthermore, notwithstanding the exclusive rights the Owner has granted to ACM, Owner shall have the right to do the following:

(i) Reuse any portion of the Work, without fee, in any future works written or edited by the Author, including books, lectures and presentations in any and all media.

(ii) Create a "Major Revision" which is wholly owned by the author

(iii) Post the Accepted Version of the Work on (1) the Author's home page, (2) the Owner's institutional repository, (3) any repository legally mandated by an agency funding the research on which the Work is based, and (4) any non-commercial repository or aggregation that does not duplicate ACM tables of contents, i.e., whose patterns of links do not substantially duplicate an ACM-copyrighted volume or issue. Non-commercial repositories are here understood as repositories owned by non-profit organizations that do not charge a fee for accessing deposited articles and that do not sell advertising or otherwise profit from serving articles.

(iv) Post an "Author-Izer" link enabling free downloads of the Version of Record in the ACM Digital Library on (1) the Author's home page or (2) the Owner's institutional repository;

(v) Prior to commencement of the ACM peer review process, post the version of the Work as submitted to ACM ("Submitted Version" or any earlier versions) to non-peer reviewed servers;

(vi) Make free distributions of the final published Version of Record internally to the Owner's employees, if applicable;

(vii) Make free distributions of the published Version of Record for Classroom and Personal Use;

(viii) Bundle the Work in any of Owner's software distributions; and

(ix) Use any Auxiliary Material independent from the Work. (x) If your paper is withdrawn before it is published in the ACM Digital Library, the rights revert back to the author(s).

When preparing your paper for submission using the ACM TeX templates, the rights and permissions information and the bibliographic strip must appear on the lower left hand portion of the first page.

The new ACM Consolidated TeX template Version 1.3 and above automatically creates and positions these text blocks for you based on the code snippet which is system-generated based on your rights management choice and this particular conference.

NOTE: For authors using the ACM Microsoft Word Master Article Template and Publication Workflow, The ACM Publishing System (TAPS) will add the rights statement to your papers for you. Please check with your conference contact for information regarding submitting your source file(s) for processing.

Please copy and paste \setcopyright{acmcopyright} before \begin{document} and please copy and paste the following code snippet into your TeX file between \begin{document} and \maketitle, either after or before CCS codes.

```
\copyrightyear{2019}
\acmYear{2019}
\acmConference[CCS '19]{2019 ACM SIGSAC Conference on Computer and Communications Security}{November 11--15, 2019}{London, United Kingdom}
\acmBooktitle{2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19), November 11--15, 2019, London, United Kingdom}
\acmPrice{15.00}
\acmDOI{10.1145/3319535.3363217}
\acmISBN{978-1-4503-6747-9/19/11}
```

If you are using the ACM Microsoft Word template, or still using an older version of the ACM TeX template, or the current versions of the ACM SIGCHI, SIGGRAPH, or SIGPLAN TeX templates, you must copy and paste the following text block into your document as per the instructions provided with the templates you are using:

CCS '19, November 11–15, 2019, London, United Kingdom
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6747-9/19/11…$15.00
https://doi.org/10.1145/3319535.3363217

NOTE: Make sure to include your article's DOI as part of the bibstrip data; DOIs will be registered and become active shortly after publication in the ACM Digital Library. Once you have your camera ready copy ready, please send your source files and PDF to your event contact for processing.

✓ A. Assent to Assignment. I hereby represent and warrant that I am the sole owner (or authorized agent of the copyright owner(s)), with the exception of third party materials detailed in section III below. I have obtained permission for any third-party material included in the Work.

☐ B. Declaration for Government Work. I am an employee of the National Government of my country and my Government claims rights to this work, or it is not copyrightable (Government work is classified as Public Domain in U.S. only)

Are any of the co-authors, employees or contractors of a National Government? ○ Yes [✓ No]

II. Permission For Conference Recording and Distribution

* Your Audio/Video Release is conditional upon you agreeing to the terms set out below.

I hereby grant permission for ACM to include my name, likeness, presentation and comments in any and all forms, for the Conference and/or Publication.

I further grant permission for ACM to record and/or transcribe and reproduce my presentation as part of the ACM Digital Library, and to distribute the same for sale in complete or partial form as part of an ACM product on CD-ROM, DVD, webcast, USB device, streaming video or any other media format now or hereafter known.

I understand that my presentation will not be sold separately as a stand-alone product without my direct consent. Accordingly, I give ACM the right to use my image, voice, pronouncements, likeness, and my name, and any biographical material submitted by me, in connection with the Conference and/or Publication, whether used in excerpts or in full, for distribution described above and for any associated advertising or exhibition.

Do you agree to the above Audio/Video Release? ○ Yes [✓ No]

III. Auxiliary Material

Do you have any Auxiliary Materials? ○ Yes [✓ No]

IV. Third Party Materials

In the event that any materials used in my presentation or Auxiliary Materials contain the work of third-party individuals or organizations (including copyrighted music or movie excerpts or anything not owned by me), I understand that it is my responsibility to secure any necessary permissions and/or licenses for print and/or digital publication, and cite or attach them below.

[✓ We/I have not used third-party material.]
○ We/I have used third-party materials and have necessary permissions.

V. Artistic Images
If your paper includes images that were created for any purpose other than this paper and to which you or your employer claim copyright, you must complete Part V and be sure to include a notice of copyright with each such image in the paper.
[✓ We/I do not have any artistic images.]
○ We/I have any artistic images.

VI. Representations, Warranties and Covenants

The undersigned hereby represents, warrants and covenants as follows:

CITED LITERATURE

1. Hossain, M. N., Milajerdi, S. M., Wang, J., Eshete, B., Gjomemo, R., Sekar, R., Stoller, S., and Venkatakrishnan, V.: SLEUTH: Real-time attack scenario reconstruction from COTS audit data. In 26th USENIX Security Symposium (USENIX Security 17), pages 487–504, Vancouver, BC, 2017. USENIX Association.

2. Milajerdi, S. M., Gjomemo, R., Eshete, B., Sekar, R., and Venkatakrishnan, V.: HOLMES: Real-time APT Detection through Correlation of Suspicious Information Flows. In Proceedings of the IEEE Symposium on Security and Privacy. IEEE, 2019.

3. Milajerdi, S. M., Eshete, B., Gjomemo, R., and Venkatakrishnan, V.: Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pages 1813–1830. ACM, 2019.

4. M. Milajerdi, S., Eshete, B., Gjomemo, R., and Venkatakrishnan, V.: Propatrol: Attack investigation via extracted high-level tasks. In International Conference on Information Systems Security. Springer, 2018.

5. Westcott, D. and Bandla, K.: APT Notes. `https://github.com/aptnotes/data`, 2018.

6. Albanesius, C.: Target Ignored Data Breach Warning Signs. `http://www.pcmag.com/article2/0,2817,2454977,00.asp`, 2014. [Online; accessed 16-February-2017].

7. Mlot, S.: Neiman Marcus Hackers Set Off Nearly 60K Alarms. `http://www.pcmag.com/article2/0,2817,2453873,00.asp`, 2014. [Online; accessed 16-February-2017].

8. King, S. T. and Chen, P. M.: Backtracking intrusions. In SOSP. ACM, 2003.

9. King, S. T., Mao, Z. M., Lucchetti, D. G., and Chen, P. M.: Enriching intrusion alerts through multi-host causality. In NDSS, 2005.

10. Adversarial tactics, techniques and common knowledge. `https://attack.mitre.org/wiki/Main_Page`.

11. Systems, N.: LOKI, free IOC scanner - Nextron Systems. `https://www.nextron-systems.com/loki/`, 2017.

12. FireEye: Redline. `https://www.fireeye.com/services/freeware/redline.html`, 2018. Accessed: 2019-04-23.

13. MANDIANT: Exposing One of China's Cyber Espionage Units. `https://www.fireeye.com/content/dam/fireeye-www/services/pdfs/mandiant-apt1-report.pdf`. Accessed: 2016-11-10.

14. Fildes, J.: Stuxnet virus targets and spread revealed. `https://www.bbc.com/news/technology-12465688`, 2011.

15. Splunk: SIEM, AIOps, Application Management, Log Management, Machine Learning, and Compliance. `https://www.splunk.com/`, 2019.

16. Logrhythm, the security intelligence company. `https://logrhythm.com/`.

17. IBM QRadar SIEM. `https://www.ibm.com/us-en/marketplace/ibm-qradar-siem`.

18. Sekar, R.: MAPRLE Internal Technical Report. Technical report, Stony Brook University, 2015.

19. Qin, X. and Lee, W.: Statistical causality analysis of infosec alert data. In RAID. Springer, 2003.

20. Ning, P. and Xu, D.: Learning attack strategies from intrusion alerts. In CCS. ACM, 2003.

21. Wang, W.: A graph oriented approach for network forensic analysis. 2010.

22. Noble, C. C. and Cook, D. J.: Graph-based anomaly detection. In Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, 2003.

23. Kemerlis, V. P., Portokalidis, G., Jee, K., and Keromytis, A. D.: Libdft: Practical Dynamic Data Flow Tracking for Commodity Systems. SIGPLAN Not., 2012.

24. Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., Le Traon, Y., Octeau, D., and McDaniel, P.: Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. SIGPLAN Not., 2014.

25. Lee, K. H., Zhang, X., and Xu, D.: High accuracy attack provenance via binary-based execution partition. In NDSS, 2013.

26. Luk, C.-K., Cohn, R., Muth, R., Patil, H., and Klauser et al., A.: Pin: building customized program analysis tools with dynamic instrumentation. In PLDI, 2005.

27. Transparent computing. `https://www.darpa.mil/program/transparent-computing`.

28. Keromytis, A. D.: Transparent computing engagement 3 data release. `https://github.com/darpa-i2o/Transparent-Computing`, 2018.

29. Newsome, J. and Song, D.: Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. 2005.

30. Xu, W., Bhatkar, S., and Sekar, R.: Taint-enhanced policy enforcement: A practical approach to defeat a wide range of attacks. In USENIX Security, 2006.

31. McColl, R. C., Ediger, D., Poovey, J., Campbell, D., and Bader, D. A.: A performance evaluation of open source graph databases. In Proceedings of the first workshop on Parallel programming for analytics applications. ACM, 2014.

32. Neo4j graph database. `https://neo4j.com/`.

33. Titan graph database. `http://titan.thinkaurelius.com/`.

34. Ediger, D., McColl, R., Riedy, J., and Bader, D. A.: Stinger: High performance data structure for streaming graphs. In High Performance Extreme Computing (HPEC). IEEE, 2012.

35. Network-x graph database. `https://networkx.github.io/`.

36. Hossain, M. N., Wang, J., Sekar, R., and Stoller, S.: Dependence preserving data compaction for scalable forensic analysis. In USENIX Security Symposium. USENIX Association, 2018.

37. Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains. `http://www.lockheedmartin.com/content/dam/lockheed/data/corporate/documents/LM-White-Paper-Intel-Driven-Defense.pdf`. Accessed: 2016-11-10.

38. Gehani, A. and Tariq, D.: Spade: support for provenance auditing in distributed environments. In Proceedings of the 13th International Middleware Conference. Springer, 2012.

39. Bates, A., Tian, D. J., Butler, K. R., and Moyer, T.: Trustworthy whole-system provenance for the linux kernel. In USENIX Security, 2015.

40. Ma, S., Zhang, X., and Xu, D.: ProTracer: Towards practical provenance tracing by alternating between logging and tainting. In NDSS, 2016.

41. Kumar, S.: Classification and detection of computer intrusions. Doctoral dissertation, PhD thesis, Purdue University, 1995.

42. Porras, P. A. and Kemmerer, R. A.: Penetration state transition analysis: A rule-based intrusion detection approach. In Computer Security Applications Conference, 1992. Proceedings., Eighth Annual, pages 220–229. IEEE, 1992.

43. CAPEC: Common Attack Pattern Enumeration and Classification. `https://capec.mitre.org/index.html`. Accessed: 2018-02-27.

44. Common vulnerability scoring system v3.0: Specification document. `https://www.first.org/cvss/specification-document`.

45. FireEye: Open IOC. `https://openioc.org`, 2018.

46. Mitre: Structured Threat Information eXpression (STIX). `https://stixproject.github.io`, 2018.

47. MISP: MISP - Open Source Threat Intelligence Platform & Open Standards For Threat Information Sharing. `https://www.misp-project.org/`, 2019. Accessed: 2019-04-23.

48. Iklody, M. A.: Default type of relationships in MISP objects. `https://github.com/MISP/misp-objects/blob/master/relationships/definition.json`, 2019. Accessed: 2019-04-23.

49. Pei, K., Gu, Z., Saltaformaggio, B., Ma, S., Wang, F., Zhang, Z., Si, L., Zhang, X., and Xu, D.: Hercule: Attack story reconstruction via community discovery on correlated log graph. In Proceedings of the 32Nd Annual Conference on Computer Security Applications, pages 583–595. ACM, 2016.

50. Symantec: Buckeye: Espionage Outfit Used Equation Group Tools Prior to Shadow Brokers Leak. `https://www.symantec.com/blogs/threat-intelligence/buckeye-windows-zero-day-exploit`, 2019.

51. Times, N. Y.: 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`, 2019.

52. Kolbitsch, C., Comparetti, P. M., Kruegel, C., Kirda, E., Zhou, X.-y., and Wang, X.: Effective and efficient malware detection at the end host. In USENIX security symposium, volume 4, pages 351–366, 2009.

53. Zong, B., Xiao, X., Li, Z., Wu, Z., Qian, Z., Yan, X., Singh, A. K., and Jiang, G.: Behavior query discovery in system-generated temporal graphs. Proceedings of the VLDB Endowment, 9(4):240–251, 2015.

54. Wagner, D. and Soto, P.: Mimicry attacks on host-based intrusion detection systems. In Proceedings of the 9th ACM Conference on Computer and Communications Security, pages 255–264. ACM, 2002.

55. Parampalli, C., Sekar, R., and Johnson, R.: A practical mimicry attack against powerful system-call monitors. In Information, computer and communications security. ACM, 2008.

56. De Nardo, L., Ranzato, F., and Tapparo, F.: The subgraph similarity problem. IEEE Transactions on Knowledge and Data Engineering, 21(5):748–749, 2009.

57. FireEye: OpenIOC Series: Investigating with Indicators of Compromise (IOCs) - Part I. `https://www.fireeye.com/blog/threat-research/2013/12/openioc-series-investigating-indicators-compromise-iocs.html`, 2013.

58. team, M. S.: APT1 Report Converstion to STIX. `https://stix.mitre.org/language/version1.0.1/samples/README.txt`, 2013. Accessed: 2019-04-23.

59. team, M. S.: FireEye Poison Evy Report Converstion to STIX. https://stix.mitre.org/language/version1.0.1/samples/README-fireeye.txt, 2013. Accessed: 2019-04-23.

60. Zhu, Z. and Dumitras, T.: Chainsmith: Automatically learning the semantics of malicious campaigns by mining threat intelligence reports. In 2018 IEEE European Symposium on Security and Privacy (EuroS&P), pages 458–472. IEEE, 2018.

61. Liao, X., Yuan, K., Wang, X., Li, Z., Xing, L., and Beyah, R.: Acing the ioc game: Toward automatic discovery and analysis of open-source cyber threat intelligence. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pages 755–766. ACM, 2016.

62. Husari, G., Al-Shaer, E., Ahmed, M., Chu, B., and Niu, X.: Ttpdrill: Automatic and accurate extraction of threat actions from unstructured text of cti sources. In Proceedings of the 33rd Annual Computer Security Applications Conference, pages 103–115. ACM, 2017.

63. STIX: STIX Visualization. https://oasis-open.github.io/cti-documentation/stix/gettingstarted.html#stix-visualization, 2019. Accessed: 2019-05-15.

64. Moran, F. N. and Villeneuve, N.: Operation DeputyDog: Zero-Day (CVE-2013-3893) Attack Against Japanese Targets. https://www.fireeye.com/blog/threat-research/2013/09/operation-deputydog-zero-day-cve-2013-3893-attack-against-japanese-targets.html, 2013. Accessed: 2019-04-19.

65. Zong, B., Raghavendra, R., Srivatsa, M., Yan, X., Singh, A. K., and Lee, K.-W.: Cloud service placement via subgraph matching. In 2014 IEEE 30th International Conference on Data Engineering, pages 832–843. IEEE, 2014.

66. Gallagher, B.: Matching structure and semantics: A survey on graph-based pattern matching. AAAI FS, 6:45–53, 2006.

67. Fan, W., Li, J., Ma, S., Tang, N., Wu, Y., and Wu, Y.: Graph pattern matching: from intractable to polynomial time. Proceedings of the VLDB Endowment, 3(1-2):264–275, 2010.

68. Cheng, J., Yu, J. X., Ding, B., Philip, S. Y., and Wang, H.: Fast graph pattern matching. In 2008 IEEE 24th International Conference on Data Engineering, pages 913–922. IEEE, 2008.

69. Zou, L., Chen, L., and Özsu, M. T.: Distance-join: Pattern match query in a large graph database. Proceedings of the VLDB Endowment, 2(1):886–897, 2009.

70. Khan, A., Wu, Y., Aggarwal, C. C., and Yan, X.: Nema: Fast graph search with label similarity. In Proceedings of the VLDB Endowment, volume 6, pages 181–192. VLDB Endowment, 2013.

71. Tong, H., Faloutsos, C., Gallagher, B., and Eliassi-Rad, T.: Fast best-effort pattern matching in large attributed graphs. In 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2007), pages 737–746. ACM, 2007.

72. Pienta, R., Tamersoy, A., Tong, H., and Chau, D. H.: Mage: Matching approximate patterns in richly-attributed graphs. In 2014 IEEE International Conference on Big Data (Big Data), pages 585–590. IEEE, 2014.

73. Solutions, G. D. F. C.: njRAT Uncovered. `https://app.box.com/s/vdg51zbfvap52w60zj0is3l1dmyya0n4`, 2013. Accessed: 2019-04-19.

74. Blog, G. D.: The Uroburos case: new sophisticated RAT identified. `https://www.gdatasoftware.com/blog/2014/11/23937-the-uroburos-case-new-sophisticated-rat-identified`, 2013. Accessed: 2019-04-19.

75. (GReAT), K. L. G. R. . A. T.: Carbanak APT: The Great Bank Robbery. `https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2018/03/08064518/Carbanak_APT_eng.pdf`, 2015. Accessed: 2019-04-19.

76. Team, C. C. S.: Operation DustySky. `https://www.clearskysec.com/wp-content/uploads/2016/01/Operation%20DustySky_TLP_WHITE.pdf`, 2016. Accessed: 2019-04-19.

77. by ESET, W.: OceanLotus: Old techniques, new backdoor. `https://www.welivesecurity.com/wp-content/uploads/2018/03/ESET_OceanLotus.pdf`, 2018. Accessed: 2019-08-12.

78. by FortiGuard Labs, T. A.: Analysis of a New HawkEye Variant. https://www.fortinet.com/blog/threat-research/hawkeye-malware-analysis.html, 2019. Accessed: 2019-08-12.

79. hasherezade: PE-Sieve: Scans a given process. recognizes and dumps a variety of potentially malicious implants (replaced/injected pes, shellcodes, hooks, in-memory patches). https://github.com/hasherezade/pe-sieve, 2018.

80. Apache: ab - Apache HTTP server benchmarking tool. https://httpd.apache.org/docs/2.4/programs/ab.html, 2019. Accessed: 2019-08-27.

81. Workbench: Jetstream2. https://browserbench.org/JetStream/index.html, 2019. Accessed: 2019-08-27.

82. EFD: HD Tune. https://www.hdtune.com, 2019. Accessed: 2019-08-27.

83. Lee, K. H., Zhang, X., and Xu, D.: Loggc: garbage collecting audit log. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, pages 1005–1016. ACM, 2013.

84. Xu, Z., Wu, Z., Li, Z., Jee, K., Rhee, J., Xiao, X., Xu, F., Wang, H., and Jiang, G.: High fidelity data reduction for big data security dependency analyses. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pages 504–516. ACM, 2016.

85. Muniswamy-Reddy, K.-K., Holland, D. A., Braun, U., and Seltzer, M. I.: Provenance-aware storage systems. In USENIX Annual Technical Conference, 2006.

86. Goel, A., Po, K., Farhadi, K., Li, Z., and de Lara, E.: The taser intrusion recovery system. SIGOPS Oper. Syst. Rev., 2005.

87. Goel, A., Feng, W. C., Maier, D., Feng, W. C., and Walpole, J.: Forensix: a robust, high-performance reconstruction system. In 25th IEEE International Conference on Distributed Computing Systems Workshops, 2005.

88. Braun, U., Garfinkel, S., Holland, D. A., Muniswamy-Reddy, K.-K., and Seltzer, M. I.: Issues in automatic provenance collection. In International Provenance and Annotation Workshop. Springer, 2006.

89. Pohly, D. J., McLaughlin, S., McDaniel, P., and Butler, K.: Hi-fi: collecting high-fidelity whole-system provenance. In ACSAC. ACM, 2012.

90. Hassan, W. U., Lemay, M., Aguse, N., Bates, A., and Moyer, T.: Towards scalable cluster auditing through grammatical inference over provenance graphs. In Network and Distributed Systems Security Symposium, 2018.

91. Ma, S., Zhai, J., Wang, F., Lee, K. H., Zhang, X., and Xu, D.: Mpi: Multiple perspective attack investigation with semantics aware execution partitioning. In 26th {USENIX} Security Symposium ({USENIX} Security 17), pages 1111–1128, 2017.

92. Ma, S., Lee, K. H., Kim, C. H., Rhee, J., Zhang, X., and Xu, D.: Accurate, low cost and instrumentation-free security audit logging for windows. In Proceedings of the 31st Annual Computer Security Applications Conference, ACSAC 2015, pages 401–410, New York, NY, USA, 2015. ACM.

93. Kwon, Y., Wang, F., Wang, W., Lee, K. H., Lee, W.-C., Ma, S., Zhang, X., Xu, D., Jha, S., Ciocarlie, G., et al.: Mci: Modeling-based causality inference in audit logging for attack investigation. In Proc. of the 25th Network and Distributed System Security Symposium (NDSS'18), 2018.

94. Ji, Y., Lee, S., Downing, E., Wang, W., Fazzini, M., Kim, T., Orso, A., and Lee, W.: Rain: Refinable attack investigation with on-demand inter-process information flow tracking. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pages 377–390. ACM, 2017.

95. Ji, Y., Lee, S., Fazzini, M., Allen, J., Downing, E., Kim, T., Orso, A., and Lee, W.: Enabling refinable cross-host attack investigation with efficient data flow tagging and tracking. In 27th {USENIX} Security Symposium ({USENIX} Security 18), pages 1705–1722, 2018.

96. Kwon, Y., Kim, D., Sumner, W. N., Kim, K., Saltaformaggio, B., Zhang, X., and Xu, D.: Ldx: Causality inference by lightweight dual execution. ACM SIGOPS Operating Systems Review, 50(2):503–515, 2016.

97. Liu, Y., Zhang, M., Li, D., Jee, K., Li, Z., Wu, Z., Rhee, J., and Mittal, P.: Towards a timely causality analysis for enterprise security. In Network and Distributed Systems Security Symposium, 2018.

98. Hassan, W. U., Guo, S., Li, D., Chen, Z., Jee, K., Li, Z., and Bates, A.: Nodoze: Combatting threat alert fatigue with automated provenance triage. In NDSS, 2019.

99. Sun, X., Dai, J., Liu, P., Singhal, A., and Yen, J.: Using bayesian networks for probabilistic identification of zero-day attack paths. IEEE Transactions on Information Forensics and Security, 13(10):2506–2521, 2018.

100. Cavallaro, L. and Sekar, R.: Taint-enhanced anomaly detection. In Information Systems Security. Springer, 2011.

101. Denning, D. E.: An intrusion-detection model. IEEE Transactions on software engineering, 1987.

102. Lunt, T. F., Tamaru, A., and Gillham, F.: A real-time intrusion-detection expert system (IDES). SRI International. Computer Science Laboratory, 1992.

103. Forrest, S., Hofmeyr, S., Somayaji, A., Longstaff, T., et al.: A sense of self for unix processes. In S&P. IEEE, 1996.

104. Gu, G., Porras, P. A., Yegneswaran, V., Fong, M. W., and Lee, W.: Bothunter: Detecting malware infection through ids-driven dialog correlation. In USENIX Security Symposium, volume 7, pages 1–16, 2007.

105. Oprea, A., Li, Z., Yen, T.-F., Chin, S. H., and Alrwais, S.: Detection of early-stage enterprise infection by mining large-scale log data. In Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on, pages 45–56. IEEE, 2015.

106. Bilge, L., Balzarotti, D., Robertson, W., Kirda, E., and Kruegel, C.: Disclosure: detecting botnet command and control servers through large-scale netflow analysis. In Proceedings of the 28th Annual Computer Security Applications Conference, pages 129–138. ACM, 2012.

107. Antonakakis, M., Perdisci, R., Lee, W., Vasiloglou, N., and Dagon, D.: Detecting malware domains at the upper dns hierarchy. In USENIX security symposium, volume 11, pages 1–16, 2011.

108. Antonakakis, M., Perdisci, R., Nadji, Y., Vasiloglou, N., Abu-Nimeh, S., Lee, W., and Dagon, D.: From throw-away traffic to bots: Detecting the rise of dga-based malware. In USENIX security symposium, volume 12, 2012.

109. Warrender, C., Forrest, S., and Pearlmutter, B.: Detecting intrusions using system calls: Alternative data models. In S&P. IEEE, 1999.

110. Sekar, R., Bendre, M., Dhurjati, D., and Bollineni, P.: A fast automaton-based method for detecting anomalous program behaviors. In S&P. IEEE, 2001.

111. Wagner, D. and Dean, D.: Intrusion detection via static analysis. In S&P. IEEE, 2001.

112. Kruegel, C., Valeur, F., and Vigna, G.: Intrusion detection and correlation: challenges and solutions. Springer Science & Business Media, 2005.

113. Feng, H. H., Kolesnikov, O. M., Fogla, P., Lee, W., and Gong, W.: Anomaly detection using call stack information. In S&P. IEEE, 2003.

114. Lee, W., Stolfo, S. J., and Mok, K. W.: A data mining framework for building intrusion detection models. In S&P. IEEE, 1999.

115. Gao, D., Reiter, M. K., and Song, D.: Gray-box extraction of execution graphs for anomaly detection. In CCS. ACM, 2004.

116. Kruegel, C. and Vigna, G.: Anomaly detection of web-based attacks. In CCS. ACM, 2003.

117. Manzoor, E., Milajerdi, S. M., and Akoglu, L.: Fast memory-efficient anomaly detection in streaming heterogeneous graphs. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 1035–1044. ACM, 2016.

118. Berlin, K., Slater, D., and Saxe, J.: Malicious behavior detection using windows audit logs. In Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security, 2015.

119. Shu, X., Yao, D., and Ramakrishnan, N.: Unearthing stealthy program attacks buried in extremely long execution paths. In CCS. ACM, 2015.

120. Ko, C., Ruschitzka, M., and Levitt, K.: Execution monitoring of security-critical programs in distributed systems: A specification-based approach. In S&P. IEEE, 1997.

121. Uppuluri, P. and Sekar, R.: Experiences with specification-based intrusion detection. In RAID. Springer, 2001.

122. Debar, H. and Wespi, A.: Aggregation and correlation of intrusion-detection alerts. In RAID. Springer, 2001.

123. Ning, P. and Xu, D.: Learning attack strategies from intrusion alerts. In CCS. ACM, 2003.

124. Qin, X. and Lee, W.: Statistical causality analysis of infosec alert data. In RAID. Springer, 2003.

125. Noel, S., Robertson, E., and Jajodia, S.: Correlating intrusion events and building attack scenarios through attack graph distances. In ACSAC. IEEE, 2004.

126. Wang, W. and Daniels, T. E.: A graph based approach toward network forensics analysis. Transactions on Information and System Security (TISSEC), 2008.

127. Zhai, Y., Ning, P., and Xu, J.: Integrating ids alert correlation and os-level dependency tracking. In International Conference on Intelligence and Security Informatics, pages 272–284. Springer, 2006.

128. Kruegel, C., Valeur, F., and Vigna, G.: Intrusion detection and correlation: challenges and solutions, volume 14. Springer Science & Business Media, 2004.

129. Wang, X., Ding, X., Tung, A. K., Ying, S., and Jin, H.: An efficient graph indexing method. In 2012 IEEE 28th International Conference on Data Engineering, pages 210–221. IEEE, 2012.

130. Giugno, R. and Shasha, D.: Graphgrep: A fast and universal method for querying graphs. In Pattern Recognition, 2002. Proceedings. 16th International Conference on, volume 2, pages 112–115. IEEE, 2002.

131. Sun, Z., Wang, H., Wang, H., Shao, B., and Li, J.: Efficient subgraph matching on billion node graphs. Proceedings of the VLDB Endowment, 5(9):788–799, 2012.

132. Gao, P., Xiao, X., Li, D., Li, Z., Jee, K., Wu, Z., Kim, C. H., Kulkarni, S. R., and Mittal, P.: {SAQL}: A stream-based query system for real-time abnormal system behavior detection. In 27th {USENIX} Security Symposium ({USENIX} Security 18), pages 639–656, 2018.

133. Gao, P., Xiao, X., Li, Z., Xu, F., Kulkarni, S. R., and Mittal, P.: {AIQL}: Enabling efficient attack investigation from system monitoring data. In 2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18), pages 113–126, 2018.

134. Pasquier, T., Han, X., Moyer, T., Bates, A., Hermant, O., Eyers, D., Bacon, J., and Seltzer, M.: Runtime analysis of whole-system provenance. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS '18, pages 1601–1616, New York, NY, USA, 2018. ACM.

135. Shu, X., Araujo, F., Schales, D. L., Stoecklin, M. P., Jang, J., Huang, H., and Rao, J. R.: Threat intelligence computing. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS '18, pages 1883–1898, New York, NY, USA, 2018. ACM.

136. Christodorescu, M., Jha, S., and Kruegel, C.: Mining specifications of malicious behavior. In Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, pages 5–14. ACM, 2007.

137. [ms-smb2]: Server message block (smb) protocol versions 2 and 3. `https://msdn.microsoft.com/en-us/library/cc246231.aspx`.

138. About the metasploit meterpreter. `https://www.offensive-security.com/metasploit-unleashed/about-meterpreter/`.

VITA

NAME Sadegh Momeni Milajerdi

EDUCATION Ph.D., Computer Science, University of Illinois at Chicago, Chicago, Illinois, 2014-2020.
M.Sc., Information Systems Engineering, Sharif University of Technology, Tehran, Iran, 2010-2012.
B.Sc., Computer Systems Engineering, Shiraz University of Technology, Shiraz, Iran, 2006-2010.

PUBLICATIONS

Milajerdi SM, Eshete B, Gjomemo R, Venkatakrishnan VN. "POIROT: Aligning Attack Behavior with Kernel Audit Records for Cyber Threat Hunting". Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS). ACM, Nov 2019.

Milajerdi SM, Gjomemo R, Eshete B, Sekar R, Venkatakrishnan VN. "HOLMES: Real-time APT Detection through Correlation of Suspicious Information Flows". Proceedings of the IEEE Symposium on Security and Privacy (S&P). IEEE, May 2019.

Milajerdi SM, Eshete B, Gjomemo R, Venkatakrishnan VN. "ProPatrol: Attack Investigation via Extracted High-Level Tasks". International Conference on Information Systems Security. Lecture Notes in Computer Science, vol 11281. Springer, Dec 2018.

Hossain MN, **Milajerdi SM**, Wang J, Eshete B, Gjomemo R, Sekar R, Stoller S, Venkatakrishnan VN. "SLEUTH: Real-time Attack Scenario Reconstruction from COTS Audit Data". Proceedings of the 26th USENIX Security Symposium. USENIX Security, 2017.

Manzoor E, **Milajerdi SM**, Akoglu L. "Fast Memory-efficient Anomaly Detection in Streaming Heterogeneous Graphs". Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2016.

Milajerdi SM, Kharrazi M. "A Composite-Metric Path Selection Technique for the Tor Anonymity Network". Journal of Systems and Software. 2015.