HERCULE: Attack Story Reconstruction via Community Discovery on Correlated Log Graph

Kexin Pei[⋆], Zhongshu Gu[‡], Brendan Saltaformaggio[†], Shiqing Ma[†], Fei Wang[†],
Zhiwei Zhang[†], Luo Si[†], Xiangyu Zhang[†], Dongyan Xu[†]

[⋆]Columbia University, kpei@cs.columbia.edu
[‡]IBM T.J. Watson Research Center, zgu@us.ibm.com
[†]Purdue University, {bsaltafo, ma229, feiwang, zhan1187, lsi, xyzhang, dxu}@cs.purdue.edu

ABSTRACT

Advanced cyber attacks consist of multiple stages aimed at being stealthy and elusive. Such attack patterns leave their footprints spatio-temporally dispersed across *many different logs* in victim machines. However, existing log-mining intrusion analysis systems typically target only a single type of log to discover evidence of an attack and therefore fail to exploit fundamental inter-log connections. The output of such single-log analysis can hardly reveal the complete attack story for complex, multi-stage attacks. Additionally, some existing approaches require heavyweight system instrumentation, which makes them impractical to deploy in real production environments. To address these problems, we present HERCULE, an automated multi-stage log-based intrusion analysis system. Inspired by graph analytics research in *social network analysis*, we model multi-stage intrusion analysis as a *community discovery problem.* HERCULE builds multi-dimensional weighted graphs by correlating log entries *across multiple lightweight logs* that are readily available on commodity systems. From these, HERCULE discovers any "attack communities" embedded within the graphs. Our evaluation with 15 well known APT attack families demonstrates that HERCULE can reconstruct attack behaviors from a spectrum of cyber attacks that involve multiple stages with high accuracy and low false positive rates.

1. INTRODUCTION

Emerging cyber attack campaigns (e.g., enterprise-wide APT) exhibit "low-and-slow" attack patterns: attackers conduct reconnaissance and move laterally within a network via many stealthy multi-stage payloads. One annual report published by FireEye [18] highlighted that attackers may reside in a victim's environment for up to 205 days before being discovered. Importantly, the covert nature of these attacks is derived from their deliberately small footprints on each affected system.

Our analysis of the attacks spanning 15 well known cyber attack families [3, 8, 9, 12, 21, 24, 28, 47–50, 61] (Section 4) shows that the stages of a single attack will often span many users, processes, and systems. Typically, the initial stage of a successful penetration is a social engineering campaign (e.g., a phishing email), watering-hole attack [3], or trojaned software or unofficial patch containing malicious payloads [9, 21, 24, 47–49, 61]. After gaining a foothold in the network, reconnaissance payloads will be deployed, include Command and Control (C&C) channels, stealing passwords, or exploiting vulnerabilities to escalate privilege. In later stages, attackers slowly move throughout the network [8] seeking opportunities to exfiltrate confidential information, eavesdrop on communications, or interrupt critical services.

Unfortunately, any footprints left by such multi-stage attack pattern are spatio-temporally dispersed across many separate logs on different victims' machines. For instance, downloading a trojaned executable may leave evidence in the web browser's log, but accessing confidential files may only be revealed in the system audit log. Further, fine-grained on-host provenance logging systems [29, 36, 40] are still rarely deployed on end-user systems. Therefore, piecing together the contextual information of each malicious footprint (scattered across many disconnected sources) still demands significant effort from cyber investigators.

To date, most existing log-based intrusion analysis and detection systems have the following three limitations: (1) Lacking the *panoramic view* required to understand the whole attack trace due to their focus on only single log types. For example, network intrusion analysis techniques [14, 51] leverage deep packet inspection or packet headers from a single network log, such as a DNS log or an HTTP proxy log. Other systems [29, 33, 34, 36, 40] perform host-based analysis on a system's audit log, such as a Windows event log. (2) The log collection process relies on logging systems that require heavyweight instrumentation, which incurs non-trivial performance overhead. For example, to achieve fine-grained system call level traces of an attack, Gu et al. [27] perform heavyweight event tracing for Windows (ETW) logging [15]. Yin et al. [64] use whole-system fine-grained tracking to detect and analyze privacy-breaching malware. Other approaches [29,30,33,34,36,46] perform static/dynamic analysis to obtain execution models or control flow graphs. While shown to be fine-grained, their substantial overhead limits their applicability to the real-world environment. (3) Too many uncorrelated alerts may be either deemed false positives or overlooked by system administrators [1]. They provide neither much actionable intelligence nor enough attack evi-

ACSAC '16, December 05 - 09, 2016, Los Angeles, CA, USA

ISBN 978-1-4503-4771-6/16/12. . . $15.00

DOI: http://dx.doi.org/10.1145/2991079.2991122

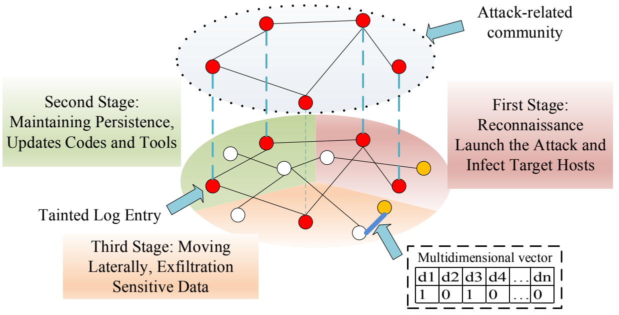

Attack-related community
Second Stage: Maintaining Persistence, Updates Codes and Tools
First Stage: Reconnaissance Launch the Attack and Infect Target Hosts
Tainted Log Entry
Third Stage: Moving Laterally, Exfiltration Sensitive Data
Multidimensional vector
d1 d2 d3 d4 ... dn
1 0 1 0 ... 0

Figure 1: Attack stages and correlated log entries.

dence. For instance, a single log entry of "user failed to login" is a *potential indicator of a compromise*, but by correlating with other alerts, such as suspicious reverse TCP activity, investigators can (manually) piece together an attacker's footprints.

In light of the above three limitations, we are motivated to focus our detection on the abundance of light-weight logs, which are more likely to be deployed in end-user systems. By correlating the many disparate entries across these logs, we aim to automatically reveal the attacker's footprints, just as an investigator would have done. We present HERCULE[1], an attack analysis framework that aims to automatically reconstruct an attack's stages and movement from multiple lightweight system logs. HERCULE is based on the observation that the attack-related log entries, triggered by the multistage payloads, have *dense and heavy-weighted* connections within themselves, but *sparse and light-weighted* connections with the benign log entries. In this way, the weighted graphs built from system logs is *similar to social networks*: people with similar interests, backgrounds, or friend circles, have closer connections to each other.

Figure 1 gives an overview of many log entries dispersed in different stages of the intrusion being correlated by multidimensional edges. The red nodes are real attack traces, and yellow nodes are the suspicious log entries not belonging to any real attacks. The white nodes represent benign log entries. The tainted log entry is marked as "attack-related," which is used later to classify the attack-related community. HERCULE is designed to discover the latent attack-related community embedded in the graph (the extracted circle in Figure 1), despite the benign and malicious log entries being highly interleaved with each other.

Based on causality analysis, HERCULE extracts various types of the correlations and construct a uniform vector representation of the connections between the log entries (Section 3.1). For example, the blue (vertical) edges in Figure 1 illustrate the multi-dimensional network edges which HERCULE assigns different feature weights, revealing the densely connected communities. We provide several versions of weight assignment in Section 3.2, which helps to increase the system performance by supervised learning and quadratic programming.

HERCULE then applies a community detection algorithm to the global graph and generates a series of community cliques. With the tainted attack-related log entry (e.g., via malware binary analysis [35,53] or website blacklisting), HERCULE reveals the attack-related communities and their interconnections based on their relationship to any tainted entries (Section 3.4).

Our work in this paper makes the following contributions:

- We propose a novel technique to model the relationship between multiple logs in the system by leveraging causality analysis without heavyweight logging or program instrumentation. HERCULE automatically generates a multi-dimensional weighted graph with potentially valuable information embedded within. Our proposed graph based representation provides a "panoramic view" of the logs generated by different system components.
- We leverage social network analysis and adapt the community detection algorithm in our weighted graph settings. We also propose several learning techniques to optimize weight assignment and increase the system performance. To the best of our knowledge, no such techniques have been adopted to date in log-based attack analysis.
- We conduct an extensive evaluation of HERCULE for the analysis of attack scenarios based on 15 real-world APT reports with diverse combinations of applications, malicious payloads, and attack methods, demonstrating the effectiveness of HERCULE.

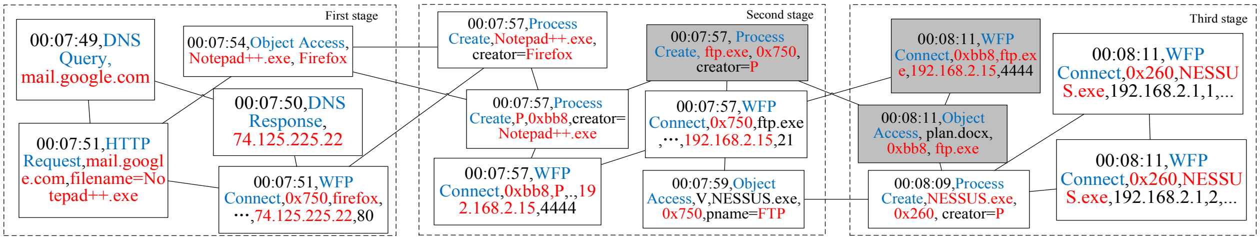
2. SYSTEM OVERVIEW

Figure 2 shows a simplified attack scenario that leaves malicious footprints across different logs. In the first stage, the user V is tempted to download a trojaned version of `Notepad++.exe` from Gmail in Firefox with a malicious payload embedded. There are three actions in the second stage: (1) V initiates the trojaned `Notepad++.exe` installation process. This causes the embedded malicious payload to open a reverse TCP connection to a remote C&C server. (2) After establishing the C&C channel, the C&C server sends the command to search for a private file `plan.docx`, attempting to collect the competitor's business plan. (3) The C&C server sends the instruction to download the `NESSUS.exe` [45] vulnerability scanner from the C&C server through FTP. In the third stage, the C&C client receives instructions to run `NESSUS.exe` to scan within the subnet and exfiltrate `plan.docx` through FTP back to the C&C server.

Figure 2 shows the connections of the attack traces *across five different logs* (those connections that investigators must manually recover). Further, besides the attack-related entries shown in Figure 2, numerous benign entries and suspicious entries (i.e., truly benign but still requiring investigation) are also recorded in the logs. The goal for HERCULE is to automatically extract and present investigators with the three attack phases and their interconnection from the many disparate log entries.

Workflow of HERCULE. Figure 3 presents the key phases and operations of HERCULE. The input to HERCULE is multiple raw logs from both network (e.g., DNS, WFP, HTTP) and system activity (e.g., Process creation, File access, Authentication). HERCULE's detection logic is specifically designed to be *log-format agnostic* and thus HERCULE can handle any input log file given that a parser

[1] HERCULE stands for **H**armful **E**pisode **R**econstruction by **C**orrelating **U**nsuspicious **L**ogged **E**vents, also as a tribute to Hercule Poirot, one of the most celebrated fictional detectives



```
00:07:49,DNS Query, mail.google.com
00:07:54,Object Access, Notepad++.exe, Firefox
00:07:50,DNS Response, 74.125.225.22
00:07:51,HTTP Request,mail.google.com,filename=Notepad++.exe
00:07:51,WFP Connect,0x750,firefox,⋯,74.125.225.22,80
00:07:57,Process Create,Notepad++.exe, creator=Firefox
00:07:57,Process Create,P,0xbb8,creator=Notepad++.exe
00:07:57,WFP Connect,0xbb8,P,,,192.168.2.15,4444
00:07:57, Process Create, ftp.exe, 0x750, creator=P
00:07:57,WFP Connect,0x750,ftp.exe,⋯,192.168.2.15,21
00:07:59,Object Access,V,NESSUS.exe, 0x750,pname=FTP
00:08:11,WFP Connect,0xbb8,ftp.exe,192.168.2.15,4444
00:08:11,Object Access, plan.docx, 0xbb8, ftp.exe
00:08:09,Process Create,NESSUS.exe, 0x260, creator=P
00:08:11,WFP Connect,0x260,NESSUS.exe,192.168.2.1,1,...
00:08:11,WFP Connect,0x260,NESSUS.exe,192.168.2.1,2,...
```

Figure 2: Parsed attack-related log entries in different types of logs (marked in blue), and the key observation to correlate the log entries (marked in red). For clarity, some log entries are slightly different from the real parsed log entries in our experiment. We also delete some edges for illustrating clear graph.

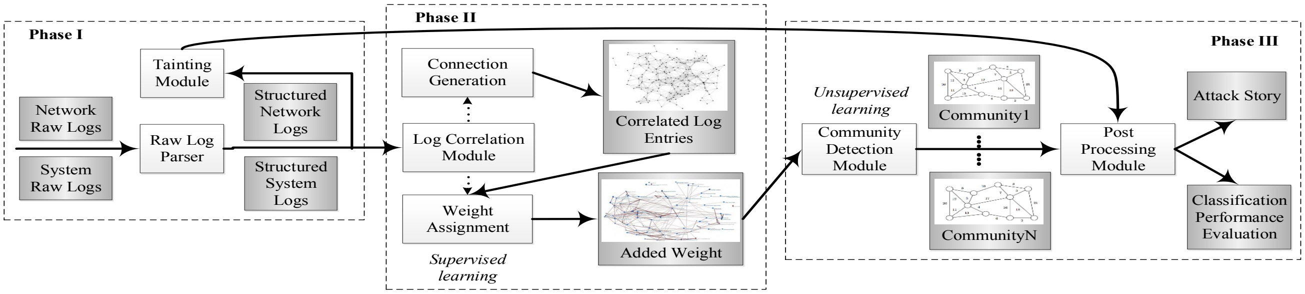


Figure 3: Workflow of HERCULE.

for its format can be supplied as a plugin. Notice that investigators would need to understand (and therefore parse) the format of these logs anyway during their investigation. Thus requiring such a parsing plugin is not a significant additional overhead. Further, parsing plugins are reusable in any future investigations of those log types. The remainder of HERCULE's operation is fully automated.

Phase I. The Raw Log Parser processes each input log entry via its parsing plugin to extract a set of predefined fields (refereed to as a data entity). Each data entity is given as input to the Log Correlation Module (Phase II) and the Tainting Module. The Tainting Module scans the data entities and (1) analyzes any suspicious executable binary appeared in the log entries that is not in a whitelist by leveraging popular malware/virus analysis platform [60, 62] or (2) scans for known malicious website accesses based on a URL blacklist to identify initial attack-related log entries. These initial attack-related log entries will subsequently be processed by the Post Processing Module.

Phase II. The Log Correlation Module consists of two submodules: *Connection Generation* and *Weight Assignment*. (1) *Connection Generation* connects any two log entries (via an unweighted multi-dimensional edge) if there exist one or more types of correlation between them (this process is detailed in Section 3.1). For example, in Figure 1, if the two nodes connected by the blue edge have the same timestamp (denoted by $d1$) or share the same process ID (denoted by $d3$), then these entries should be correlated. (2) *Weight Assignment* assigns weight on each edge. Note that there are multiple ways to assign weights to multi-dimensional edges, in which one of the supervised learning techniques has proven to achieve the best result. We present the details of weight assignment in Section 3.2.

Phase III. The Community Detection Module takes the correlated weighted graph as input and outputs all detected communities to the Post Processing Module. Starting from any log entries tainted in Phase I, the Post Processing Module classifies the communities that contain tainted entries as malicious and the others as benign. It then outputs the reconstructed attach phases (temporally ordered actions or communities) and their interconnection from the attack-related community. Section 3.4 presents the detailed design of HERCULE's community detection.

3. SYSTEM DESIGN

Log parsing is an essential step to transform the raw data into data entities before applying any learning model. Thus far, we have implemented log parsing plugins for the logs shown Table 1. To capture attack footprints projected across various logs, we select different sets of logs based on the host OS platforms. For example, on Linux, we choose the Syslogd authentication log, while on Windows we make use of the Windows filtering platform (WFP) log which records the inbound and outbound network connections the processes makes. For each log, raw log parser extracts pre-defined fields that capture representative information of each log entry. We summarize these in Table 2.

3.1 Connection Generation

The input to the Connection Generation sub-module is the parsed data entities. The output is the unweighted, undirected, and multi-dimensional graph which is built from the intra-log and inter-log correlation. Analogous to a social network, we treat each log entry as an individual and each edge dimension as one type of relationship between two individuals. Formally, in the unweighted n-dimensional network $G = (V, E, D)$ where V is a set of nodes, E is a set of edges and D is a set of dimensions, G forms a $|V| \times |V| \times |D|$

Table 1: Logs used in Section 4.

#	Logs	Provider
L1	DNS	Tshark
L2	WFP connect	Auditd
L3	HTTP	Firefox
L4	Process create	Auditd
L5	Object access	Auditd
L6	Authentication	Syslogd

Table 2: Fields Correlated Across Logs.

Field	Logs	Description
timestamp	L1-L6	event timestamp
q_domain	L1	DNS quiered domain name
r_ip	L1	DNS resolved ip address
pid	L2, L4, L5	base-16 process id
ppid	L4	base-16 parent process id
pname	L2, L4, L5, L6	process name
h_ip	L2	host IP address
h_port	L2	host port number
d_ip	L2	destination IP address
d_port	L2	destination port number
type	L3	request/response
get_q	L3	absolute path of GET
post_q	L3	absolute path of POST
res_code	L3	response code
h_domain	L3	host domain name
referer	L3	referer of requested URI
res_loc	L3	location to redirect
acct	L5	principle of this access
objname	L5	object name
info	L6	Authentication information

Table 3: Features Described by Each Edge.

D	Feature
d_1	Δ(*u.timestamp, v.timestamp*)$<t$
d_2	*u.pid=v.pid*
d_3	*u.d_ip=v.d_ip*
d_4	*u.d_port=v.d_port*
d_5	*u.referer=v.referer*
d_6	*u.host=v.host*
d_7	*u.referer=v.host*
d_8	*u.host=v.referer*
d_9	*u.ppid=v.ppid*
d_{10}	*u.ppid=v.pid*
d_{11}	*u.pid=v.ppid*
d_{12}	*u.objname=v.objname*
d_{13}	*u.pname=v.pname*
d_{14}	*u.r_ip=v.d_ip*
d_{15}	*u.d_ip=v.r_ip*
d_{16}	*u.q_domain=v.h_domain*
d_{17}	*u.h_domain=v.q_domain*
d_{18}	*u.q_domain=v.referer*
d_{19}	*u.referer=v.q_domain*
d_{20}	*u.q_domain=v.res_loc*
d_{21}	*u.res_loc=v.q_domain*
d_{22}	*u.get_q=v.pname*
d_{23}	*u.pname=v.get_q*
d_{24}	*u.get_q=v.objname*
d_{25}	*u.objname=v.get_q*
d_{26}	*u.pname=v.objname*
d_{27}	*u.objname=v.pname*
d_{28}	*u.r_ip=v.h_ip*
d_{29}	*u.h_ip=v.r_ip*

3-dimensional boolean matrix M. $M_{i,j,k} = 1$ indicates there exists a correlation dimension k between log entry node i and log entry node j, otherwise $M_{i,j,k} = 0$. Each $e \in E$ consists of $(n{+}2)$-tuples $(i, j, d_1, d_2, \cdots, d_n)$ with $i, j \in V$ and $d_1, \cdots, d_n \in D$.

Feature Selection. In HERCULE, for each pair of log entries, denoted as nodes u and v, which have one or more types of relationships, we connect them with multi-dimensional edge e. The dimensions of the edge is denoted as a uniform 29-feature vector $\vec{v} = [d_1\ d_2\ \cdots\ d_{29}]^T$, in which the binary value of each dimension d_k represents the existence of the k-th type of relationship between u, v. If u, v can be correlated by k-th relationship, $d_k = 1$, otherwise $d_k = 0$.

The intuition behind each of the 29 features to capture potential causally-related log entries (summarized in Table 3) is as follows. For two log entries u and v:

- d_1 models the time difference between u and v. The user can customize the threshold t (between two timestamps of u and v) to determine whether two log entries are temporal correlated. We show the detection performance result of choosing different t in Section 4.

- d_2 and d_{13} captures the relationship if u, v share the same process id or process name. This feature is intended to capture the explicit/implicit correlations from process creation and process operations.

- d_3 and d_4 demonstrate whether u and v share the same destination IP or port when they make outbound connection requests. The intuition is based on the observation that u and v (two network requests) are highly correlated if they communicate to the same IP address.

- d_5 to d_8 belong to the features that capture the correlation of u and v within HTTP log. They model the causality relations of web page browsing events. Several attack reports in Section 4 show that the initial stage of an attack often involves a sequence of URL redirection. Thus, we aim to recover a complete browsing traces that are related to the attack story.

- d_9 to d_{11} model the process creation correlations. For instance, we observe that many malicious operations are done by a shell process so that we need to trace back the parent or predecessor process of that shell to recover more attack traces.

- d_{12} checks if u and v access to the same object. It is based on the observation that some malicious process creates the malicious executable (logged in u), and another process executes the executable (logged in v). u and v belong to the same attack traces and hence $d_{12} = 1$ denotes that u and v are causally related.

- d_{14}, d_{15}, d_{28}, and d_{29} model how outbound/inbound network requests correlate to DNS queries. The intuition is based on the fact that most C&C client/server communications leverage DNS service to resolve dynamically changing server IP addresses [51].

- d_{16} to d_{21} capture the correlation between DNS query behavior and web page browsing behavior by examining the equality of several related fields. We seek to include in the attack story of DNS resolving query and the corresponding browsing events on websites.

- d_{22} to d_{27} model how web browsing behaviors reflected in HTTP log correlate to system-level behaviors, e.g., a user may download a malicious executable from the web and execute it locally. It can also spawn new processes, access objects and make network connections.

It is possible that there exist more features beyond those in Table 3. HERCULE is extensible in that users can add or remove features to customize the system.

Also, note that these features can lead to false-positive connections in our graphs. This is because any local noise in these features will be discarded during community detection, which focuses exclusively on global patterns (communities) within the resulting global graph.

With defined features on each edge, we build the connections for each log entry from all the input logs. The input to the connection generation algorithm are the features selected in Table 3 and all parsed structured logs. The algorithm only generates edges with at least one feature vector value being non-zero. The algorithm iterates all possible log pairs that capture both intra-log correlations and inter-log correlations.

3.2 Weight Assignment

Once the correlated log graph has been constructed, we use a community detection algorithm to detect communities. Formally, given two community clusters of nodes A and B in an unweighted multi-dimensional graph G where A denotes attack-related log entries and B represents benign entries: optimally, we expect to get $|e_A| \gg |e_{AB}|$ and $|e_B| \gg |e_{AB}|$ where $|e_A|$, $|e_B|$, and $|e_{AB}|$ denote the number of edges in cluster A, B, and between A and B, respectively. Note that these notations are valid throughout all this section.

However, it is observed that in the unweighted multi-dimensional graph generated in Section 3.1, there exist cases of log entries that belong to the attack but have numerous connections with benign entries ($|e_A| < |e_{AB}|$). This case significantly reduces the effectiveness of any community detection algorithm that aims to maximize the intra-cluster density $|e_A|$, $|e_B|$ and minimizes inter-cluster density $|e_{AB}|$ (we present the details of the algorithm in Section 3.4). On the other hand, we make the observation that the values of the feature dimensions between $\vec{e_{AB}}$ and $\vec{e_A}$, $\vec{e_B}$ ($\vec{e_A}$, $\vec{e_B}$ and $\vec{e_{AB}}$ denote the edge vector of e_A, e_B and e_{AB}, respectively) are significantly different. This implies that we can distinguish e_A, e_B from e_{AB} by their feature vector.

Inspired by this observations, weight assignment algorithm is therefore proposed to deal with the above circumstances. Generally, the algorithm tries to assign different weights to edges that have different edge feature values so that the inequality still holds $w_A \cdot |e_A| > w_{AB} \cdot |e_{AB}|$ (w_A is weight assigned for edges in e_A and w_{AB} is weight assigned for edges in e_{AB}). More specially, the algorithm tries to "learn" a global weight vector $\vec{\alpha}$ that can be applied on each edge. Thus, the following equation still holds:

$$\begin{aligned}\sum\nolimits_{e\in e_A}\sum_{i=1}^{k}\alpha_i\cdot e_i > \sum\nolimits_{e\in e_{AB}}\sum_{i=1}^{k}\alpha_i\cdot e_i\\ \sum\nolimits_{e\in e_B}\sum_{i=1}^{k}\alpha_i\cdot e_i > \sum\nolimits_{e\in e_{AB}}\sum_{i=1}^{k}\alpha_i\cdot e_i\end{aligned} \tag{1}$$

where k is the number of dimensions of edge vector and e_i is the i-th value of edge vector $\vec{e}$.

Intuitively, we can assign the dot product of the weight vector and edge vector $w = \sum_{i=1}^{k}\alpha_i e_i$ as the weight on each edge to construct a weighted graph. However, most of the learning algorithms that we apply in weight assignment output $\vec{\alpha}$ such that the dot product has no bound on the value ($w = \sum_{i=1}^{k}\alpha_i e_i) \in \mathbb{R}$, which might generate negative weight $w < 0$ and violate the requirement of input for the weighted graph community detection algorithm. Consequently, we leverage a sigmoid function S to map the dot product to bounded real number range $[0, 1]$ as our finalized weight assignment on each edge: $w = S(\sum_{i=1}^{k}\alpha_i \cdot e_i) = \frac{1}{1+e^{-\sum_{i=1}^{k}\alpha_i\cdot e_i}}$

We then transform the graph into a weighted graph WG. For evaluation of the weight assignment algorithm, we define our *training phase* and *testing phase* as following: Given n unweighted graphs $G_1, G_2, \cdots, G_n$, for each l ($l \in [1, n]$), (1) the *training phase* of the weight assignment looks for a best assignment weight vector $\vec{\alpha_k}$ for G_1, $\cdots$, G_{l-1}, G_{l+1}, $\cdots$, G_n; (2) the *testing phase* takes the dot product of weight vector $\vec{\alpha_l}$ and all edge vectors $\vec{e}$ to generate a weighted graph WG_l. Then, the Community Detection Module takes WG_l as input and outputs communities for Post Processing Module. Essentially, this training/testing process adopts the leave-one-out strategy.

We have also built different algorithms for weight assignment for comparison. The first does not use any learning algorithm, the second and third leverage existing supervised learning techniques that outperform the first, and the fourth is based on quadratic optimization that has the best performance results. We use the quadratic optimization algorithm as our finalized version of weight assignment in HERCULE, and compare their results quantitatively in Section 4.

Feature Weight Summation. As a baseline solution, this algorithm treats each feature of an edge with the same "importance": $\alpha_i = 1$, $i \in [1, k]$ where k denotes the number of features. To hold Equation 1, this algorithm depends on the assumption that edges in A or B have more correlation types than edges between A and B. More specifically, this algorithm assumes the edge vectors of $\vec{e_A}$ and $\vec{e_B}$ has more 1's than edge vectors of $\vec{e_{AB}}$. From the results presented in Section 4, we find that the performance is not ideal. The reason is that there exist cases when two edge vectors, one from e_A, e_B and the other from e_{AB}, (1) are not distinguishable by their number of 1's of their vector values, but differ in type of features where the 1 reside, such as two vectors [1 0 1] and [0 1 1], or even worse, (2) the edge vector in e_A or e_B has less 1's in the vector values than those in e_{AB}.

Motivated by the discussed limitations, we adapt two learning approaches (Logistic regression and SVM), which assign different "importance" on the edge features.

Logistic Regression. Logistic regression [44] can be applied in our weight assignment as we abstract the weight assignment as a classification problem. We want to learn the global weight $\vec{\alpha}$ that helps to classify edge vectors into one class e_A, e_B and another class e_{AB}. Suppose there are m training edges, denoted $E = x_i, y_i, i \in [1, m]$ where x_i is i-th edge vector $\vec{e_i}$, $y_i = 1$ if $e_i \in e_{AB}$ and $y_i = 0$ if $e_i \in e_A$ or $e_i \in e_B$. In training, we construct a prediction function, which leverages logistic function g: $h_{\vec{\alpha}}(x_i) = g(\alpha^T x_i) = \frac{1}{1+e^{-\vec{\alpha}x_i}}$ where $h_{\vec{\alpha}}(x)$ denotes the probability that $e_i \in e_{AB}$ ($y_i = 1$): $P(y_i = 1|x_i, \vec{\alpha}) = h_{\vec{\alpha}}(x_i)$ and $P(y_i = 0|x_i, \vec{\alpha}) = 1 - h_{\vec{\alpha}}(x_i)$. Then we should minimize the cost function in log likelihood format: $-\frac{1}{m}[\sum\limits_{i=1}^{m} y_i \log h_{\vec{\alpha}}(x_i) + \sum\limits_{i=1}^{m}(1-y_i)\log(1-h_{\vec{\alpha}}(x_i))]$. The minimization problem can be solved by using gradient descent. Please refer to [44] for more details of the algorithm.

SVM. Another classification solution for learning the weight vector α is an SVM [19]. Suppose there are m training edges, denoted $E = x_i, y_i, i \in [1, m]$ where x_i is i-th edge vector $\vec{e_i}$, $y_i = 1$ if $e_i \in e_{AB}$ and $y_i = -1$ if $e_i \in e_A$ or

$e_i \in e_B$. The purpose of the SVM is to learn a weight vector $\vec{\alpha}$, which can accurately distinguish e_{AB} from e_A, e_B. We use the soft margin version; the detailed formulation can be found in [19].

Quadratic Programming. The above two classification learning algorithms look for a decision boundary that classifies edges into two types (e_A, e_B vs. e_{AB}). However, their output weight vectors are not the global optimum, which are not guaranteed to maximize the weight assigned to edge e_A and e_B, and minimize the weight assigned to edge e_{AB}.

Therefore, we design and develop a new solution, which transforms the weight assignment as a quadratic optimization problem. Adapted from Equation 1, our target function is:

$$\begin{aligned} \max_{\vec{\alpha}} \quad & \sum\nolimits_{e \in e_A} \sum_{i=1}^{k} \alpha_i \cdot e_i + \sum\nolimits_{e \in e_B} \sum_{i=1}^{k} \alpha_i \cdot e_i \\ & -\lambda \sum\nolimits_{e \in e_{AB}} \sum_{i=1}^{k} \alpha_i \cdot e_i - \frac{1}{2} \vec{\alpha}^T \cdot \vec{\alpha} \\ & s.t. \quad 0 \leq \vec{\alpha}^T e \leq 1 \end{aligned} \tag{2}$$

λ is the trade-off parameter to balance between the first two terms and the third term in the target function. $\frac{1}{2}\vec{\alpha}^T \cdot \vec{\alpha}$ is the regularizer to avoid the overfitting problem. The target function is convex and the output weight vector $\vec{\alpha}$ is theoretically global optimum. As we constrained the optimization by $0 \leq \vec{\alpha}^T e \leq 1$, we do not leverage a sigmoid function for this algorithm to map the dot product $\vec{\alpha}^T e$ to $[0, 1]$ again.

3.3 DFS Propagation

Prior to coming up with the community detection approach, we designed a heuristic algorithm that neither uses any learning techniques nor community detection, which we term "DFS propagation". We note that without the knowledge of any low-level program execution analysis or fine-grained log analysis, we cannot capture the accurate causality relationship between log entries. Thus in this algorithm, we conservatively treat any log entry t as malicious if there exists a path from a tainted attack-related start point s to t, which we define *s can propagate to t*. The input to this algorithm is the tainted entry point v and the graph generated from Log Correlation Module. It then uses Depth-First-Search (DFS) to recover all log entries that can be propagated from v. From the results shown in the Section 4, we found the performance of DFS Propagation is not ideal. Therefore, we propose a more robust method using community detection in the following section.

3.4 Community Detection

Considering the large number of nodes in the graph, we need to select a time-efficient community detection algorithm to extract communities from the large weighted correlated-log graph. There are multiple unsupervised learning techniques for community detection. We choose to use the Louvain method considering its efficient handling of large networks [13].

At the beginning of this algorithm, each node in our weighted graph represents a community. We denote $A_{v,w}$ as the weight between node v and node w, $k_v = \sum_w A_{v,w}$ as the sum of the weights connected to the node w, c_v as the community to which the node v is assigned, the δ-function $\delta(i, j) = 1$ if i = j and $\delta(i, j) = 0$ otherwise. Modularity is defined as:

$$\begin{aligned} Q &= \frac{1}{2m} \sum_{v,w} \left[A_{v,w} - \frac{k_i k_j}{2m} \right] \delta(c_v, c_w) \\ & where \;\; m = \frac{1}{2} \sum_{v,w} A_{v,w} \end{aligned} \tag{3}$$

Modularity has a value between -1 and 1, which measures the degree of the density of the connections within communities compared to connections between communities in the graph.

After the initialization of communities, we repeat the Louvain method in two phases to greedily optimize the local modularity as the algorithm progresses. For each node v, the algorithm removes v from its own community and moves it into the community C of each neighbor w of v. Then the algorithm evaluates changes in modularity and places v in the community that has the largest modularity gain. If the largest gain is negative, the node v is not moved and placed in its original community. The modularity gain is calculated:

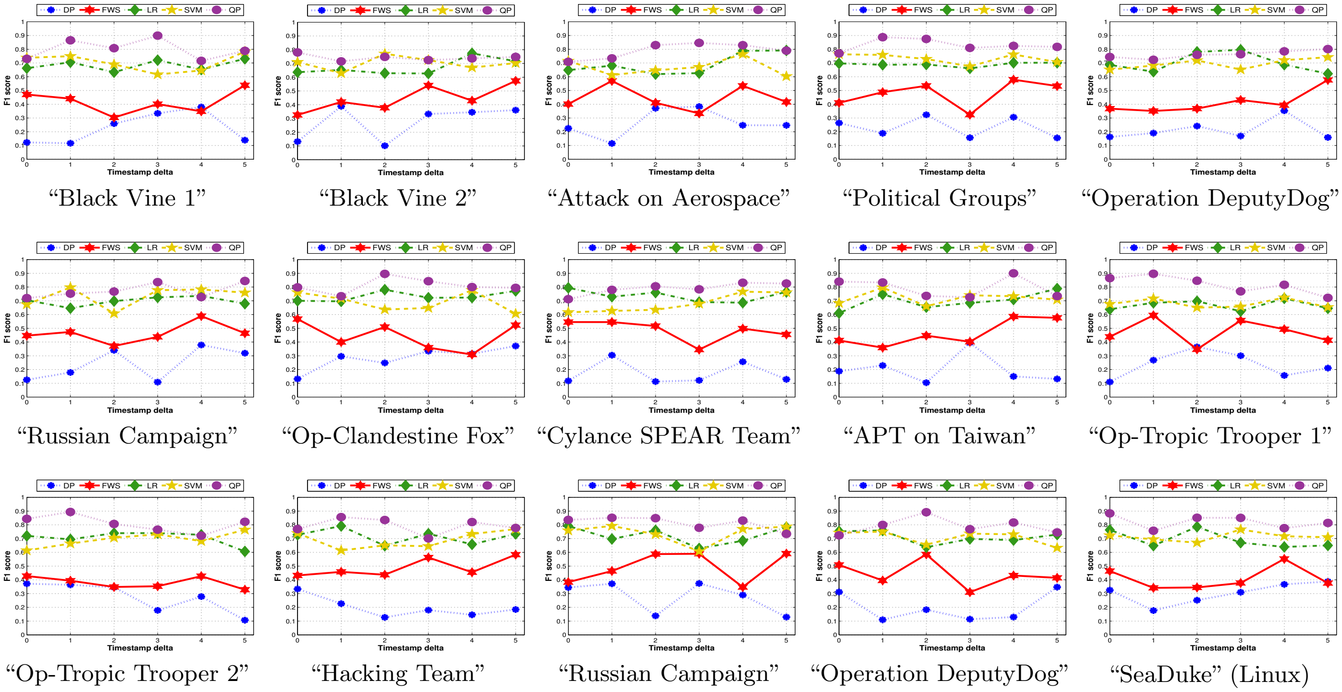
$$\begin{aligned} \Delta Q = & \left[\frac{\sum_{in} + k_{v,in}}{2m} - \left(\frac{\sum_{tot} + k_v}{2m} \right)^2 \right] \\ & - \left[\frac{\sum_{in}}{2m} - \left(\frac{\sum_{tot}}{2m} \right)^2 - \left(\frac{k_v}{2m} \right)^2 \right] \end{aligned} \tag{4}$$

$\sum_{in}$ is the sum of the weights of the edges inside C where the v is moved to, $k_v = \sum_w A_{v,w}$ denotes the sum of the weights connected to the node w, $\sum_{tot}$ is the sum of the weights of the connections to all the nodes in the community C, $k_{v,in}$ is the sum of the weights of the links to node v in community C and m is the total weights of all the edges in the graph. This process is the first phase, and it is applied repeatedly and sequentially to all nodes until no modularity increase occurs. Then the second phase begins once the algorithm reaches the local optimum of modularity. In the second phase, the algorithm aggregates all of the nodes in the same community into one node and builds a new network. The links between nodes within the same community are then represented by self-linked edges on the new community node. The second phase ends once the network reconstruction finishes, and the first phase then starts the next iteration on the new network.

4. EVALUATION

Implementation. We have implemented HERCULE in Python and use Matlab for the learning algorithms. We leverage Python's implementation of the Louvain method package `python-louvain` [39] for community detection. The weight assignment algorithms are implemented using Matlab `quadprog` [43] for quadratic programming, `LIBSVM` [38] for soft margin SVM and Matlab `glmfit` [42] for logistic regression.

Our evaluation environment consists of: (1) A Windows victim system running on a machine with an Intel Core i5-3570 3.40 GHz CPU, 4GB RAM and Windows 7 Ultimate Service Pack 1 64-bit operating system. (2) A Linux victim system runs on the machine with Intel Core i5-4200M 2.50 GHz CPU, 4GB RAM and Ubuntu 14.04.1 LTS 64-bit operating system. (3) The attacker runs on a machine with Intel Core i5-4200M 2.50 GHz CPU, 4GB RAM and the Kali Linux 64-bit operating system. The attacker's machine also serves different roles, such as the C&C server, the FTP server for downloading attack tools, the samba server for sharing files, and Apache server for hosting malicious websites.



"Black Vine 1" "Black Vine 2" "Attack on Aerospace" "Political Groups" "Operation DeputyDog"

"Russian Campaign" "Op-Clandestine Fox" "Cylance SPEAR Team" "APT on Taiwan" "Op-Tropic Trooper 1"

"Op-Tropic Trooper 2" "Hacking Team" "Russian Campaign" "Operation DeputyDog" "SeaDuke" (Linux)

Figure 4: DFS Propagation vs Different Weight Assignment Algorithms with Community Detection.

Table 4: Log Size.

Log	Size (KB)	# Entries
DNS	60	450
WFP	16,723	662
HTTP	150,650	3,016
Process	5,886	233
Object	5268	247
ETW	2,926,425	1,462,526

Table 5: Two-Week Log Size.

Log	Size (MB)
DNS	7.0875
WFP	1933.9
HTTP	17795.5
Process	695
Object	622.2

4.1 Logging Overhead

Table 4 (left) shows a summary of the logging overhead of different logging providers. The logging process records the log entry while the user is routinely working on the computer, without knowing that some of his/her operations have already triggered the malicious activity. We obtain these results by logging for 10000 seconds, and each time we launch a particular attack (Table 6) we roll back using a previously saved image of the virtual machine. Then we average the result of the 15 test cases. Compared with ETW logging with stack-walk mode turned on (shown in the last gray row of Table 4), the logs we use in HERCULE (the other rows in Table 4) are more lightweight in all measurements. This motivates the need for HERCULE's community detection based only on the abundance of light-weight logs (as heavy-weight logging is seldom deployed on end-user systems).

4.2 Weight Assignment

Figure 4 shows the detection performance of DFS propagation (blue) and the combinations of different weight assignment algorithms together with Louvain community detection (other colors), in which (1) the purple line denotes quadratic programming, (2) the yellow line denotes SVM (support vector machine), (3) the green line represents logistic regression, (4) the red line stands for feature vector summation algorithm, and (5) the blue line denotes DFS propagation.

Instead of utilizing accuracy, we quantify the performance of classifiers using F_1 score (y-axis). Let the number of true positives, false positives, true negatives, and false negatives be tp, fp, tn, and fn, respectively, w.r.t, a classifier. Precision is defined to measure the portion of actual attack-related entries in all predicted attack-related entries (identified attack-related community). $precision = \frac{tp}{tp+fp}$. Recall measures the number of attack-related entries that are correctly classified as attack-related out of the total number of actual attack-related entries. $recall = \frac{tp}{tp+fn}$. F_1 score is the harmonic mean of precision and recall $F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$. An ideal classifier has F_1 metric close to 1 which implies that both precision and recall are close to 1.

As discussed in Section 3.2, we adopt "leave-one-out" in the training/testing process of weight assignment. After the log entries have been generated, we obtain the labels of the log entries by manually comparing with the attack activity[2]. The x-axis in Figure 4 includes choices of different timestamp delta threshold t in constructing the temporal correlation. Figure 4 demonstrates the average outperformance of Quadratic programming pair with community detection (purple line) over other algorithms. This motivates our decision to use quadratic programming as our weight assignment algorithm. The results in Table 6 are based on the weight assignment algorithm that uses quadratic programming.

4.3 Attack Story Reconstruction

Emulated Attacks. Based on existing attack reports, we emulated 15 real-world APT attacks [3, 9, 12, 21, 24, 28, 47–

[2] Note that HERCULE did not have access to this ground truth for its detection. We obtained it manually for evaluation of HERCULE's accuracy.

Table 6: Evaluation Results on Emulated Attacks.

APT Keyword and Report	Initial Tactics	CVE	Post Exploitation	Target	Acc	FP
"Black Vine 1" [12]	Watering hole	2012-4792	Keylogger	Win	0.846	0.0012
"Black Vine 2" [12]	Email attachment	2014-0322	Exfiltrate files	Win	0.834	0.0023
"Attack on Aerospace" [3]	Watering hole	2015-5122	Network sweeping	Win	0.810	0.0018
"Political Groups" [61]	Email google drive links	2014-4114	Exfiltrate files	Win	0.886	0.0013
"Op-DeputyDog" [48]	iframe background running	2013-3893	Escalate privilege	Win	0.877	0.0024
"Russian Campaign" [49]	Controlled Website	2015-3043	Download backdoor	Win	0.833	0.0023
"Op-Clandestine Fox" [47]	Email compromised website	2014-1776	Rename payload	Win	0.857	0.0026
"Cylance SPEAR Team" [21]	Email attachment	2012-0158	Browsing files	Win	0.826	0.0016
"APT on Taiwan" [9]	Email attachment	N/A	Rename payload	Win	0.819	0.0010
"Op-Tropic Trooper 1" [50]	Email attachment	2010-3333	Download tools	Win	0.812	0.0006
"Op-Tropic Trooper 2" [50]	Email attachment	2012-0158	Keylogger	Win	0.863	0.0090
"Hacking Team" [28]	Email with file link	2015-5119	Download backdoor	Win	0.859	0.0058
"Russian Campaign" [49]	Email attachment	2008-5499	Download backdoor	Linux	0.850	0.0017
"Op-DeputyDog" [48]	Email compromised website	N/A	Brute force Login	Linux	0.899	0.0060
"SeaDuke" [24]	Email trojaned-ware	N/A	Add bad user	Linux	0.874	0.0012
Two weeks	"Political Groups" + "APT on Taiwan" + "Cylance SPEAR"			Win	0.736	0.0126

50, 61], which feature a variety of initial compromise tactics, types of CVE exploited, malicious payloads, and post exploitation activities to evaluate our approach (Table 6). Note that in our emulated environment, we also include a dedicated user to perform regular (benign) activities, such as browsing websites or playing music. To demonstrate HERCULE's scalability, we also conducted a two-week-long experiment that contains mostly normal user activities, which are generated by the dedicated user, mixed with three APT attacks (as listed in the last, gray row of Table 6). The dedicated user performs normal behavior such as browsing website, watching videos, and downloads and executes files without adequate awareness of the suspiciousness. The logs sizes from this two-week experiment are summarized in Table 4 (right). As some reports do not disclose a complete attack trace, we recreate any missing attack stages (not discussed in the APT reports) with modern attack approaches borrowed from similar stages discussed in the other APT reports. Hence the emulated attacks, though based on the APT reports, leverage the mixed strategies that are thus not restricted to any specific attack vectors, but provide general test cases on which HERCULE can evaluate.

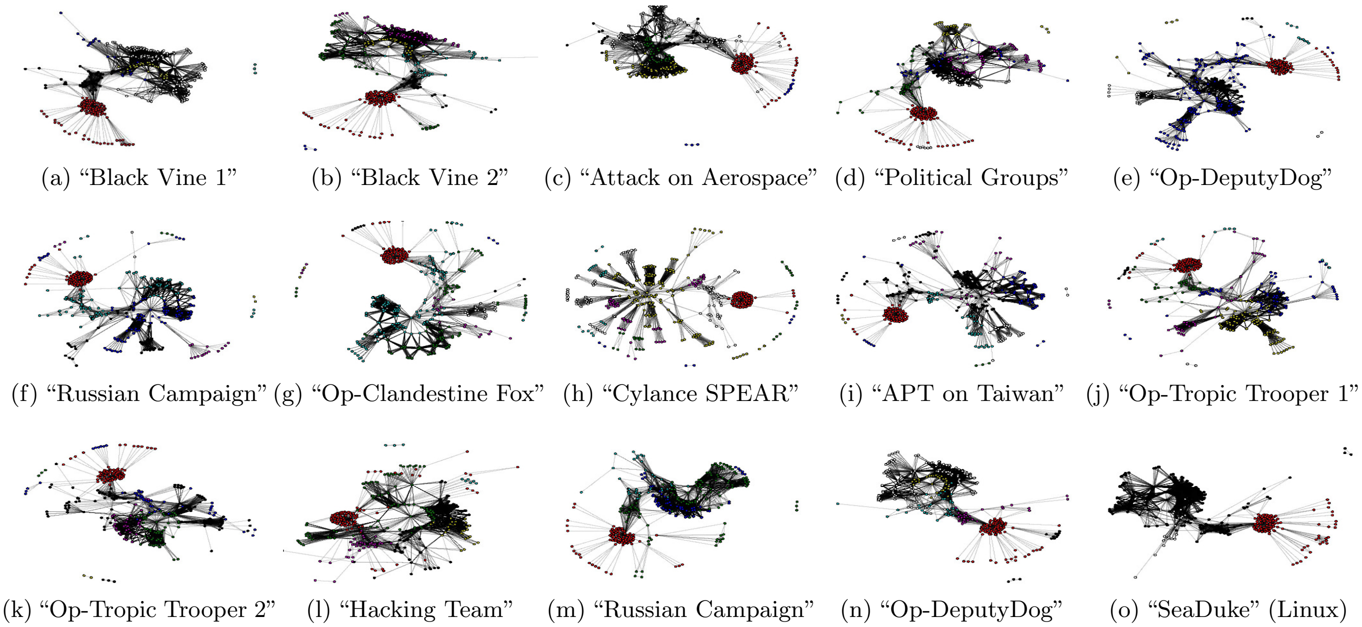
The detection performance of all 16 attack scenarios is shown in the last two columns in Table 6. We classify the log entries within any identified community as malicious and the remainders as benign. We manually obtained and inspected the ground truth from the APT reports when emulating the attacks. The accuracy is the portion of the true results (correctly classified as benign or malicious) in the total test samples. The false positive rate measures the number of entries that are incorrectly classified as attack-related out of total actual benign entries. We note that the false negative rate is not negligible given the high accuracy and extremely low false positive rate. Since our work focuses on reducing the false positive rate by correlating PIOC (Section 1) and reconstruct the attack-related *as complete as possible*, we give more tolerance on false negatives and leave the task of reducing false negative in our future work.

We also visualize the community detection output in Figure 5 (the two-week experiment graph is too dense to be shown). The community marked in red in each network denotes the identified group of malicious log entries, and we randomly mark the other communities with different colors. We assign one community with only one color. From these figures, we can see that the community identified as attack-related is well-clustered, which has dense connections within themselves but sparse connections with other communities. Note that the edge weights are not explicitly reflected in the figures. There are some red nodes that are not densely connected to the major red node community. However, this problem is dealt with, as discussed in Section 3.2, by training a larger weight assignment between this sparsely connected red nodes such that they can be still grouped into the same community.

In the remainder of this section, we present two illustrative case studies on the attack stories and their related log entries. Admittedly, there remains a certain amount of manual analysis to reconstruct attack story from the extracted community. However, the amount of work is greatly reduced for cyber investigator as the suspicious log entries are narrowed and grouped together. Furthermore, some key fields within the attack-related log entries, such as time stamp, are fairly useful indicators of the temporal attack sequence.

Story 1 ("Political Groups"): This attack exploits CVE-2014-4114, a vulnerability in the OLE package manager [61]. The attacker can leverage this vulnerability to create a PowerPoint presentation in which the OLE package manager loads (1) a malicious payload/executable camouflaged as a ".gif" file and (2) a malicious .inf file that renames the ".gif" to ".gif.exe" and runs the executable. We use MS14-060 Microsoft Windows OLE Package Manager Code Execution in Metasploit to generate the malicious ppsx file. For crafting the spear-phishing email, we utilize Social-Engineer Toolkit (SET).

In the first stage, the attacker crafts a simple message in the spear-phishing email urging the targeted recipient V to download a file from Google Drive. The google drive link points to `biography.ppsx` which is a PowerPoint view-only slideshow. The file does not display properly on Google Drive. Therefore, the victim may want to download and open the file on his/her computer. The use of Google Docs is potentially evidence of attackers' evolving tactics in reaction to the increasing number of users' checking the integrity of email attachments. In the second stage, once double-clicked in the host system, the slide show is played by victim without crashing the program or producing any other signs that something is wrong with the file. At the same time, the OLE package manager automatically copies two files, `gzrs.gif`



(a) "Black Vine 1" (b) "Black Vine 2" (c) "Attack on Aerospace" (d) "Political Groups" (e) "Op-DeputyDog"

(f) "Russian Campaign" (g) "Op-Clandestine Fox" (h) "Cylance SPEAR" (i) "APT on Taiwan" (j) "Op-Tropic Trooper 1"

(k) "Op-Tropic Trooper 2" (l) "Hacking Team" (m) "Russian Campaign" (n) "Op-DeputyDog" (o) "SeaDuke" (Linux)

Figure 5: Community Distribution (The Red Nodes Denote the Malicious Community).

and `JPih.inf` from the attacker's Samba share folder. The `JPih.inf` is then accessed by OLE package manager. Based on the instructions in `JPih.inf`, the `GzRs.gif` is renamed to `GzRs.gif.exe` and executed. In the third stage, the process `GzRs.gif.exe` makes a reverse TCP connection to the C&C server's IP address 192.168.2.15. The attacker gains the reverse shell on the victim system and subsequently browses different folders and files. After a while, the attacker locates the `plan.txt` and initiates the FTP client in command line to upload `plan.txt` to the IP address 192.168.2.15.

We extract the latent correlations of the log entries generated from the attack flow via HERCULE with accuracy 88.6% and false positive rate 0.0013 (Table 6). Figure 5d shows the attack-related well-clustered entries that have a few links with other benign entries. This case reflects the difficulty of reconstructing these APT attack traces when relying only on analysis of individual, separate logs. For instance, in single log analysis using only WFP connection log, we can identify a continuous suspicious C&C-like communication from `GzRs.gif.exe` to the IP address, which relates to a sequence of commands to browse the files. However, we know neither how the malicious backdoor is delivered nor what this backdoor does in the background.

Story 2 ("Operation DeputyDog" on Linux host): The initial compromise tactic in this attack leverages an iframe by adding it into a website's HTML code that automatically loads the attacker's page and prompts for a Firefox update [48]. We adapt this strategy to infect the Linux host by creating a trojaned executable in ELF format with a reverse HTTPS payload embedded. We leverage Msfvenom to generate and encode different types of shellcode in the Metasploit framework to create the trojaned Firefox-update ELF file. We use the encoding scheme shikata_ga_nai (a polymorphic XOR additive feedback encoder) to encode the reverse HTTPS payload 21 times to achieve obfuscation. We do not use any CVE in the infection. We also use SET to generate the spear-phishing email. For reproducing the iframe-loaded page, we leverage the BeEF framework to hook the browser by launching an XSS attack through a malicious web page. When the victim opens the malicious page in the browser, the attacker can inject an iframe to prompt users for updating the Firefox.

The campaign starts with a phishing email including a link shortened by Bitly, a URL shortening service, pointing to a web page with the malicious iframe loaded. In this case, the victim is tempted to browse the web page in the phishing email and download and run the trojaned executable firefoxupdate prompted from the webpage. The payload embedded in firefoxupdate creates a reverse HTTPS backdoor connecting to the C&C server. The C&C server then sends a command to launch a dictionary attack. By brute-force "sudo", the attacker attempts to gain root privilege. After failing for a few times, the attacker changes strategies: he/she sends instructions to the victim to start an FTP client and download the keylogger shell script, which leverages the xmodmap in Linux, from the FTP server hosted on the same IP address. At the same time, the attacker also dumps the hash file from /etc/passwd to crack the password offline. In case that either key logging or the offline cracking succeeds to gain the password of the root, the attacker can "sudo -i" to root right away. Then a new user with the name "bad" is added with root privilege, the malicious payload is renamed and is set to start automatically whenever system boot up.

Table 6 shows the accuracy 87.7% and false positive rate 0.0024. From Figure 5n, we can easily identify the well-clustered attack-related community.

5. DISCUSSION

Logging Process. As discussed in Section 4, currently we capture the logs at different levels on a single host. However in a large-scale enterprise network, it is possible to obtain logs from different hosts. In the future, we could extend

HERCULE to build a larger (i.e., cross-domain) weighted graph by creating connections of logs from different machines. We could thus gain a deeper understanding of the attack propagation behaviors by reconstructing more sophisticated stories.

Log Correlation. Though shown to be lightweight according to the size of log files (Table 4), our Log Correlation Module is restricted by the coarse-grained 29 correlation rules. There are log entries that are inherently related, but too implicit to be connected by our temporal and semantic causal analysis. We intend to add more implicit causal relations between the log events by combining the dynamic analysis of the program behavior with the learning module, to perform finer-grained attack analysis.

6. RELATED WORK

Log-Based Attack Analysis. Log analysis techniques have been studied for many years. Assuming that attackers unintentionally leave footprints in the host system, defenders can discover evidence of the attack by interpreting and analyzing logs.

A large number of attack detection approaches are based on analyzing network logs to detect anomalous network behaviors. DNS log data is widely used for detecting malicious domain names (e.g., [6, 7]). Oprea et al. [51] apply a belief propagation algorithm to DNS log data or web proxy data to identify suspicious domains. Bailey et al. [10] use raw event logs to extract higher-level malware behavior — via state changes rather than system calls — and use hierarchical clustering to group malwares that have similar classes of behaviors. Gu et al. [26] leverage network traffic to identify the coordination dialog that occurs during a malware infection. HERCULE also employs network logs to detect and track attacks, but unlike these efforts, HERCULE focuses on correlating the multiple stages of an attack across network and system layer logs.

Many attack analysis techniques handle system activity logs. For instance, system audit logs have been used by several research efforts (e.g., Kim et al. [32], Goel et al. [25], King et al. [34], Kim et al. [33], and Newsome et al. [46]). These approaches adopt backward and forward tracking to find the entry point of an attack and determine the damages inflicted to the victim systems. They still focus on individual logs and thus might not be able to detect attack stages which occur in logs outside of their focus.

To obtain system-call granularity logging of an attack's execution, LEAPS [27] performed heavyweight event logging via Event Tracing for Windows (ETW) [15]. Behavior-based detections, using higher-level abstractions of malicious logic, also rely on logging system calls or fine-grained program/kernel execution to capture intrinsic malicious behaviors [16, 20]. Dolan-Gavitt et al. [23] leverage the hypervisor to log all virtual address accessing for the signature generation that can be applied to analyze/detect malware. Lee et al. [36] perform static and dynamic analysis on applications to identify unit-level execution instrumentation points. While such approaches allow for fine-grained causal analysis of attacks, their substantial runtime overhead might limit their applicability in real-world production environments. HERCULE leverages *lightweight* logs, extracted from generally available logging mechanisms, to reveal an attack's multiple phases.

Statistical-based Intrusion Analysis. Bilge et al. propose Disclosure [11], which identifies several groups of features from NetFlow records (a single log type) to distinguish C&C channels from benign traffic using a Random Forest algorithm. Rossow et al. develop PROVEX [58], which leverages network-based statistical learning methods to detect bots in encrypted C&C channels. West et al. [63] combine blacklist histories with spatial context and trained an SVM learning model to classify spam emails. Peisert et al. [52] employ instance-based learning to conduct forensic analysis of function call sequences. Abad et al. [2] introduce the idea of multiple log correlation for intrusion detection. Cipriano et al. [17] develop Nexat, which includes its designed supervised machine learning algorithm to learn from attackers' past behavior and predict the future actions of the attacker. Kapravelos et al. [31] leverage different dimension reduction and clustering algorithms to perform efficient code similarity matching to detect evasion attempts of malicious JavaScript. Perl et al. [55] propose VCCFinder that trains an SVM classifier on C/C++ code databases to detect vulnerabilities based on features from GitHub metadata. Piro et al. [56] aim at detecting Sybil attacks in mobile ad hoc networks, which involves several learning algorithms to analyze the relationship between the radio-equipped nodes. Lee et al. [37] develop WARNINGBIRD, which leverages logistic regression with support vector classification (SVC), to detect suspicious URLs in the Twitter stream. Maggi et al. [41] apply the "special form of Hidden Markov Model" to detect intrusions using the system calls sequences. Oprea et al. [51] adapt belief propagation on large-scale DNS logs to detect malicious domains related to the C&C activity. Zhang et al. [65, 66] predict triggering relationships between network events to classify and pinpoint malicious network events based on the validity of the triggering relations. HERCULE combines unsupervised community detection to extract highly correlated attack-related communities with supervised learning for edge weight optimization that both isolates malicious/benign events and also provides insight into the multiple stages of an attack within communities.

McBoost [54] improves the scalability of malware analysis by combining multiple classifiers to classify packed/unpacked and malicious/benign executables. Polychronakis et al. [57] use general machine learning techniques to score the suspiciousness of URLs for further processing. These techniques complement HERCULE as they can be used to detect the initially tainted log entries.

Dash et al. [22] develop DroidScribe which leverages Conformal Prediction and an SVM to deal with sparse behavior profiles in classifying Android malware. Shu et al. [59] propose constrained agglomerative clustering algorithms to uncover the attack traces in long system call sequences. Almgren et al. [4] explore the application of active learning on the intrusion detection. Besides the out-performance of active learning over traditional learning algorithms, their work shows a significant training data reduction required by the active learners. Amann et al. [5] deploy a summary statistics framework to support user-defined statistics in anomaly detection. These works deal with one or more specific problems in anomaly/attack detection that are complementary to HERCULE. In contrast, HERCULE focuses on the reconstruction of attack phases, without special attention to issues about the learning techniques such as sparse behavior profiles, reducing training data size, or long attack traces.

7. CONCLUSION

We have presented HERCULE, an automated multi-stage intrusion analysis system, to reconstruct a complete, intuitive, and human-understandable attack story from multiple correlated logs, without the burden of heavyweight logging process. Our extensive evaluation on a wide spectrum of different real-world APT attacks targeting both Linux and Windows hosts shows the effectiveness of our technique in reconstructing the attack stories including different infection strategies, the vulnerabilities exploiting methods, and post exploitation operations, with high accuracy and low false positive rate.

8. ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive comments. This research was supported, in part, by DARPA under contract FA8650-15-C-7562, NSF under award 1409668, ONR under contract N000141410468, and Cisco Systems under an unrestricted gift. Any opinions, findings, and conclusions in this paper are those of the authors only and do not necessarily reflect the views of our sponsors.

9. REFERENCES

[1] Target ignored hacker alarms as crooks took 40m credit cards. www.theregister.co.uk/2014/03/14/ target_failed_to_act_on_security_alerts/, 2014.

[2] C. Abad, J. Taylor, C. Sengul, W. Yurcik, Y. Zhou, and K. Rowe. Log correlation for intrusion detection: A proof of concept. In *Proceddings of the 19th Annual Computer Security Applications Conference*, 2003.

[3] Watering hole attack on aerospace firm exploits cve-2015-5122 to install isspace backdoor. researchcenter.paloaltonetworks.com/2015/07/ watering-hole-attack-on-aerospace-firm-exploits-cve-2015-5122-to-install-isspace-backdoor/, 2015.

[4] M. Almgren and E. Jonsson. Using active learning in intrusion detection. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop*, 2004.

[5] J. Amann, S. Hall, and R. Sommer. Count me in: Viable distributed summary statistics for securing high-speed networks. In *Proceedings of the 17th International Symposium on Research in Attacks, Intrusions and Defenses*, 2014.

[6] M. Antonakakis, R. Perdisci, W. Lee, N. Vasiloglou II, and D. Dagon. Detecting malware domains at the upper dns hierarchy. In *Proceedings of the 20th USENIX Security Symposium*, 2011.

[7] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou II, S. Abu-Nimeh, W. Lee, and D. Dagon. From throw-away traffic to bots: Detecting the rise of dga-based malware. In *Proceedings of the 21st USENIX Security Symposium*, 2012.

[8] Apt kill chain - part 5 : Access strenghtening and lateral movements. http://blog.airbuscybersecurity.com/post/2014/11/ APT-Kill-chain-Part-5-%3A-Access-Strenghtening-and-lateral-movements, 2014.

[9] Dtl-06282015-01: Apt on taiwan - insight into advances of adversary ttps. blog.dragonthreatlabs.com/2015/07/ dtl-06282015-01-apt-on-taiwan-insight.html, 2015.

[10] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario. Automated classification and analysis of internet malware. In *Proceedings of the 10th International Symposium on Research in Attacks, Intrusions and Defenses*, 2007.

[11] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel. Disclosure: Detecting botnet command and control servers through large-scale netflow analysis. In *Proceedings of the 28th Annual Computer Security Applications Conference*, 2012.

[12] Black vine: Formidable cyberespionage group targeted aerospace, healthcare since 2012. www.symantec.com/ connect/blogs/black-vine-formidable-cyberespionage-group-targeted-aerospace-healthcare-2012, 2015.

[13] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008.

[14] K. Borders and A. Prakash. Web tap: detecting covert web traffic. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, 2004.

[15] I. Buch and R. Park. Improve debugging and performance tuning with etw. *MSDN Magazine*, 2007.

[16] D. Canali, A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu, and E. Kirda. A quantitative study of accuracy in system call-based malware detection. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, 2012.

[17] C. Cipriano, A. Zand, A. Houmansadr, C. Kruegel, and G. Vigna. Nexat: A history-based approach to predict attacker actions. In *Proceedings of the 27th Annual Computer Security Applications Conference*, 2011.

[18] Complimentary Report. www2.fireeye.com/WEB-2015RPTM-Trends.html, 2015.

[19] C. Cortes and V. Vapnik. Support-vector networks. *Mach. Learn.*, 1995.

[20] J. R. Crandall, G. Wassermann, D. A. de Oliveira, Z. Su, S. F. Wu, and F. T. Chong. Temporal search: Detecting hidden malware timebombs with virtual machines. In *The 2006 ACM Sigplan Notices*, 2006.

[21] Cylance spear team: A threat actor resurfaces. blog.cylance.com/spear-a-threat-actor-resurfaces, 2015.

[22] S. K. Dash, G. Suarez-Tangil, S. Khan, K. Tam, M. Ahmadi, J. Kinder, and L. Cavallaro. Droidscribe: Classifying android malware based on runtime behavior. *Mobile Security Technologies (MOST)*, 2016.

[23] B. Dolan-Gavitt, A. Srivastava, P. Traynor, and J. Giffin. Robust signatures for kernel data structures. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, 2009.

[24] Duke apt group's latest tools: cloud services and linux support. www.f-secure.com/weblog/archives/00002822.html, 2015.

[25] A. Goel, W.-c. Feng, W.-c. Feng, and D. Maier. Automatic high-performance reconstruction and recovery. *Computer Networks*, 2007.

[26] G. Gu, P. A. Porras, V. Yegneswaran, M. W. Fong, and W. Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *Proceedings of the 16thth USENIX Security Symposium*, 2007.

[27] Z. Gu, K. Pei, Q. Wang, L. Si, X. Zhang, and D. Xu.

Leaps: Detecting camouflaged attacks with statistical learning guided by program analysis. In *Proceedings of the 45th IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015.

[28] Unpatched flash player flaw, more pocs found in hacking team leak. blog.trendmicro.com/trendlabs-security-intelligence/unpatched-flash-player-flaws-more-pocs-found-in-hacking-team-leak/, 2015.

[29] X. Jiang, A. Walters, D. Xu, E. H. Spafford, F. Buchholz, and Y.-M. Wang. Provenance-aware tracing of worm break-in and contaminations: A process coloring approach. In *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, 2006.

[30] M. G. Kang, S. McCamant, P. Poosankam, and D. Song. Dta++: Dynamic taint analysis with targeted control-flow propagation. In *Proceedings of the 18th Network and Distributed System Security Symposium*, 2011.

[31] A. Kapravelos, Y. Shoshitaishvili, M. Cova, C. Kruegel, and G. Vigna. Revolver: An automated approach to the detection of evasive web-based malware. In *Proceedings of the 22nd USENIX Security Symposium*, 2013.

[32] C. H. Kim, J. Rhee, H. Zhang, N. Arora, G. Jiang, X. Zhang, and D. Xu. Introperf: transparent context-sensitive multi-layer performance inference using system stack traces. In *Proceedings of the 2014 ACM International Conference on Measurement and Modeling of Computer Systems*, 2014.

[33] T. Kim, X. Wang, N. Zeldovich, and M. F. Kaashoek. Intrusion recovery using selective re-execution. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation*, 2010.

[34] S. T. King, Z. M. Mao, D. G. Lucchetti, and P. M. Chen. Enriching intrusion alerts through multi-host causality. In *Proceedings of the 12th Network and Distributed System Security Symposium*, 2005.

[35] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X. Zhou, and X. Wang. Effective and efficient malware detection at the end host. In *Proceedings of the 18th Conference on USENIX Security Symposium*, 2009.

[36] K. H. Lee, X. Zhang, and D. Xu. High accuracy attack provenance via binary-based execution partition. In *Proceedings of the 20th Network and Distributed System Security Symposium*, 2013.

[37] S. Lee and J. Kim. Warningbird: Detecting suspicious urls in twitter stream. In *Proceedings of the 19th Network and Distributed System Security Symposium*, 2012.

[38] Libsvm. https://www.csie.ntu.edu.tw/~cjlin/libsvm/.

[39] Louvain community detection api in python. http://perso.crans.org/aynaud/communities/api.html.

[40] S. Ma, X. Zhang, and D. Xu. Protracer: Towards practical provenance tracing by alternating between logging and tainting. In *Proceedings of the 2016 Network and Distributed System Security Symposium*, 2016.

[41] F. Maggi, M. Matteucci, and S. Zanero. Detecting intrusions through system call sequence and argument analysis. *IEEE Transactions on Dependable and Secure Computing*, 2010.

[42] Matlab glmfit. http://www.mathworks.com/help/stats/glmfit.html.

[43] Matlab quadratic programming. http://www.mathworks.com/help/optim/ug/quadprog.html.

[44] P. McCullagh and J. A. Nelder. *Generalized linear models*. CRC press, 1989.

[45] Nessus vulnerability scanner. http://www.tenable.com/products/nessus-vulnerability-scanner.

[46] J. Newsome and D. Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In *Proceedings of the 12th Network and Distributed System Security Symposium*, 2005.

[47] Operation clandestine fox now attacking windows xp using recently discovered ie vulnerability. www.fireeye.com/blog/threat-research/2014/05/operation-clandestine-fox-now-attacking-windows-xp-using-recently-discovered-ie-vulnerability.html, 2014.

[48] Operation deputydog: Zero-day (cve-2013-3893) attack against japanese targets. www.fireeye.com/blog/threat-research/2013/09/operation-deputydog-zero-day-cve-2013-3893-attack-against-japanese-targets.html, 2013.

[49] Operation russiandoll: Adobe & windows zero-day exploits likely leveraged by russia's apt28 in highly-targeted attack. www.fireeye.com/blog/threat-research/2015/04/probable_apt28_useo.html, 2015.

[50] Operation tropic trooper: Old vulnerabilities still pack a punch. blog.trendmicro.com/trendlabs-security-intelligence/operation-tropic-trooper-old-vulnerabilities-still-pack-a-punch/, 2015.

[51] A. Oprea, Z. Li, T.-F. Yen, S. Chin, and S. Alrwais. Detection of early-stage enterprise infection by mining large-scale log data. In *Proceedings of the 45th IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015.

[52] S. Peisert, M. Bishop, S. Karin, and K. Marzullo. Analysis of computer intrusions using sequences of function calls. *IEEE Transactions on Dependable and Secure Computing*, 2007.

[53] F. Peng, Z. Deng, X. Zhang, D. Xu, Z. Lin, and Z. Su. X-force: Force-executing binary programs for security applications. In *Proceedings of the 23rd USENIX Security Symposium*, 2014.

[54] R. Perdisci, A. Lanzi, and W. Lee. Mcboost: Boosting scalability in malware collection and analysis using statistical classification of executables. In *Proceedings of the 24th Annual Computer Security Applications Conference*, 2008.

[55] H. Perl, S. Dechand, M. Smith, D. Arp, F. Yamaguchi, K. Rieck, S. Fahl, and Y. Acar. Vccfinder: Finding potential vulnerabilities in open-source projects to assist code audits. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015.

[56] C. Piro, C. Shields, and B. N. Levine. Detecting the sybil attack in mobile ad hoc networks. In *Proceddings of the 2nd International Conference on Security and Privacy in Communication Networks*, 2006.

[57] M. Polychronakis and N. Provos. Ghost turns zombie: Exploring the life cycle of web-based malware. 2008.

[58] C. Rossow and C. J. Dietrich. Provex: Detecting botnets with encrypted command and control channels.

In *Proceedings of the 10th SIG SIDAR Conference on Detection of Intrusions and Malware & Vulnerability Assessment*. 2013.

[59] X. Shu, D. D. Yao, and N. Ramakrishnan. Unearthing stealthy program attacks buried in extremely long execution paths. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security*, 2015.

[60] D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. G. Kang, Z. Liang, J. Newsome, P. Poosankam, and P. Saxena. Bitblaze: A new approach to computer security via binary analysis. In *Proceedings of the 4th International Conference on Information Systems Security*, 2008.

[61] Targeted attacks against tibetan and hong kong groups exploiting cve-2014-4114. citizenlab.org/2015/06/targeted-attacks-against-tibetan-and-hong-kong-groups-exploiting-cve-2014-4114/, 2015.

[62] V. Total. Virustotal-free online virus, malware and url scanner, 2012.

[63] A. G. West, A. J. Aviv, J. Chang, and I. Lee. Spam mitigation using spatio-temporal reputations from blacklist history. In *Proceedings of the 26th Annual Computer Security Applications Conference*, 2010.

[64] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda. Panorama: capturing system-wide information flow for malware detection and analysis. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, 2007.

[65] H. Zhang, D. D. Yao, and N. Ramakrishnan. Detection of stealthy malware activities with traffic causality and scalable triggering relation discovery. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, 2014.

[66] H. Zhang, D. D. Yao, N. Ramakrishnan, and Z. Zhang. Causality reasoning about network events for detecting stealthy malware activities. *computers & security*, 2016.