MART: Targeted Attack Detection on a Compromised Network

Jack W. Stokes*, Himanshu Chandola*, Christian Seifert*, and Tim Burrell†
*Microsoft Corporation, One Microsoft Way, Redmond, WA 98052, USA
†Microsoft Corporation, Montpellier House, Montpellier Drive, Cheltenham, Gloucestershire, GL50 1TY, UK

***Abstract*—Targeted attacks are a significant problem for governmental agencies and corporations. We propose a MinHash-based, targeted attack detection system which analyzes aggregated process creation events typically generated by human keyboard input. We start with a set of malicious process creation events, and their parameters, which are typically generated by an attacker remotely controlling computers on a network. The MinHash algorithm allows the system to efficiently process hundreds of millions of events each day. We propose the weighted squared match similarity score for targeted attack detection which is more robust to mimicry and NOOP attacks than the weighted Jaccard index. We demonstrate that the system can detect several confirmed targeted attacks on both a small dataset of 1,473 computers as well as a large network of over 230 thousand computers. In the first case, the proposed system detects a similar, but separate attack while in the latter, intrusion activity is detected at large-scale.**

I. Introduction

In this paper, we propose a new system to detect targeted attacks, once an adversary has successfully compromised a network, based on the similarity of aggregated process creation events and their parameters to those from previously discovered intrusions. The detection of targeted attacks [1] is one of the most important problems facing the computer security community today: the ramifications of a government, corporate, or non-governmental organization network intrusion can be catastrophic. We define targeted attacks as the broad class of intrusions where an organization or government agency is specifically targeted for the express purpose of exfiltrating some type of data or information. This data may include credit card or social security numbers for financial gain by the attackers. Other types of targeted attacks carried out by more sophisticated actors, including nation state sponsored groups, may target and gain access to an organization or government's network for the purpose of extracting other types of data such as classified documents, product or military design documents, and email or patient records. These threat actors often establish multiple redundant communication channels allowing the attacker to remain on the network even when analysts discover and disable compromised computers and accounts. In the initial stages of a targeted campaign, the attacker uses techniques such as spearphishing or social engineering [2] to entice a targeted user into installing malware containing a backdoor for communications and a keylogger to harvest their credentials. The attack may involve previously unseen zero days or exploits that target recently discovered, but potentially un-patched vulnerabilities. Once the attacker opens a beachhead on the network using the infected computer and stolen credentials, he then uses lateral movement techniques such as pass-the-hash [3] to move from one computer to the next in order to explore the network and establish redundant computer and user accounts for command and control.

While there are a number of possible ways to search for targeted attacks, one key strategy is to detect suspicious *human activity* on the network. Based on the observation that targeted attackers sometimes reuse a common set of techniques and tools to explore the network [4], we designed MART, a weighted squared *m*atch, t*ar*geted a*t*tack detection system. MART monitors the process creation events, including the parameters, that someone inputs to the computer from an operating system command shell to infer *human* intent instead of detecting the tools directly. In most cases, malicious commands are input to the computer using a remote connection, but the input could also be logged directly from the keyboard in the case of an insider threat. MART monitors this activity by logging the Windows 4688 Process Creation Events which allows it to determine the parent process and the individual processes and arguments launched from the parent process. Typically, the parent process is a Windows command shell or a Windows PowerShell. MART then constructs an aggregated *ProcessTrace* from these individual processes (i.e. commands) started from the parent process and searches for correlations of these process traces with those from confirmed targeted attacks. An example of a *ProcessTrace* in shown in Table I. Detection is based on correlation of the patterns of shell commands used, the process names, and types and actual parameters employed during the attacks. Starting from a set of malicious *ProcessTraces*, MART then searches for *ProcessTraces* from all computers on the network exhibiting similar user behavior. Searching for similar files or web pages based on a collection of malicious seeds was earlier proposed in [5], [6], [7]. We demonstrate that this approach is also effective for detecting targeted attacks.

For our system, we consider two types of similarity measures. The first finds similar human activity based on the unweighted or weighted Jaccard index [5], [6], [7]. The Jaccard index, however, is susceptible to mimicry and NOOP attacks [8] where benign or useless commands are executed to prevent detection. As a result we propose the weighted squared match (WSM) similarity score, and its unweighted equivalent, to make MART more robust to these attacks. We also propose to use the MinHash algorithm [9], which is

C:\Windows\System32\ipconfig.exe /all
C:\Windows\System32\ping.exe -4 -n 1 <targeted hostname>
C:\Windows\System32\net.exe use <targeted hostname> USER:<user name> <PW>
C:\Users\userA\malware7.exe param1 param2
C:\Windows\System32\ftp.exe -v <destination hostname>

TABLE I: Example *ProcessTrace*.

detailed in Section II, to efficiently filter *ProcessTracces* which are not similar. In Section IV, we investigate eight alternative systems which utilize either tokens or N-Grams as features, as well as no feature weighting and feature weighting with term frequency, inverse document frequency (TFIDF) weights [9]. MART's feature representations are detailed in Section II. Additional related work is provided in Section VI.

To demonstrate the practicality and effectiveness of MART, we implement it on a large-scale MapReduce system and evaluate it using two datasets containing *confirmed* targeted attacks collected from an anonymized organization. In some cases, Microsoft customers provide their Windows security logs to the Microsoft Threat Intelligence Center (MSTIC) as part of a planned joint engagement in order to help manually identify targeted attack activity on their network. The organization gave permission for the researchers to study two collections of their data logs, which are described in Section III, and report the findings to the security community as long as the organization's name is anonymized.

MART is designed to identify similar activity from persistent actors based on correlated human activity. In addition, MART can also detect attacks from new actors who employ similar techniques. Results in Section IV demonstrate that by starting with four malicious *ProcessTraces*, MART can identify additional instances of the first actor's activity as well as discover a trace from a second, *unrelated* attack. Results also show that MART can detect targeted attack activity at extremely large scale from a set of over 379 million *ProcessTraces* collected from over 230,000 computers. A summary of the contributions of this work includes:

- A MinHash-based system is proposed to detect targeted attacks via human activity from process creation logs.
- A new weighted, squared match similarity score is introduced to minimize the effectiveness of mimicry and NOOP attacks.
- The system is implemented on a MapReduce framework to handle large-scale data.
- An analysis of two datasets containing three confirmed targeted attacks is performed to evaluate the proposed system.
- Results indicate that the system can help identify similar activity from the same targeted attacker. In addition, MART is also able to discover an unrelated, but similar attack.

II. MART System Overview

In this section, we describe the proposed MART targeted attack detection system which is designed to automatically detect similar attacks and allow local system administrators to search for new attacks not identified by their other defensive systems. The MART system is depicted at a high-level in Figure 1. Windows security events 4688 corresponding to process creation are first collected from a set of computers and stored in log files. These events are then processed and aggregated to form a *ProcessTrace* dataset containing process paths, names and parameters being executed by each user. Identical or known *ProcessTraces* are then filtered from the raw input stream to reduce the computational and storage costs. After an efficient MinHash-based algorithm is used to significantly reduce the number of potential pairs of *ProcessTraces* that need to be considered, a similarity score is computed for each of the remaining *ProcessTraces* using the newly proposed weighted squared match score or the weighted Jaccard index. If the similarity score is greater than a prescribed threshold, network access is blocked and the user's account is disabled automatically. Optionally, an analyst manually investigates the top scores to identify potential intrusions. If a new or previously known intrusion with an unknown *ProcessTrace* is discovered, it is appended to the set of confirmed intrusion *ProcessTraces* to improve future detections. We next discuss details of some of these system components.

Feature Extraction: We consider two types of features in this paper including tokens and N-grams which are designed to handle the variations and intentional obfuscations employed by the attacker.

Token Features: The goal of the token features is to create a set of increasingly specific features which will hopefully aid detection with TFIDF weighting. These features have higher precision, but may result in lower recall. We also want to evaluate token features as a baseline for the N-Gram features described next. To construct the token features, each *ProcessTrace* string is split on whitespaces, "\", "/" and "-". The union of the result forms the initial set of token features for the *ProcessTrace*. Examples of token features from the first two commands in Table I include {C:, Windows, System32, ipconfig.exe, all, ping.exe, 4, n, 1, <targeted hostname>}. To address the problem where files can be moved to different directories or hard or logical drives, we construct an additional set of features for the paths. This second feature set is formed by concatenating the lowest level subdirectory in the path and the process name to create another token feature, then adding the next lowest subdirectory to this feature to create the next token feature, and so on. The final token feature is the entire path including the drive or network share. Again referring to the first two commands in Table I, the additional path token features are {System32\ipconfig.exe, Windows\System32\ipconfig.exe, C:\Windows\System32\ipconfig.exe, System32\ping.exe, Windows\System32\ping.exe, C:\Windows\System32\ping.exe}. By forming features in this manner, we attempt to generate increasingly precise features while still allowing for some form of path obfuscation in case the higher level subdirectory changes but the lower level subdirectory and process name remain the same. These two groups of token features are then

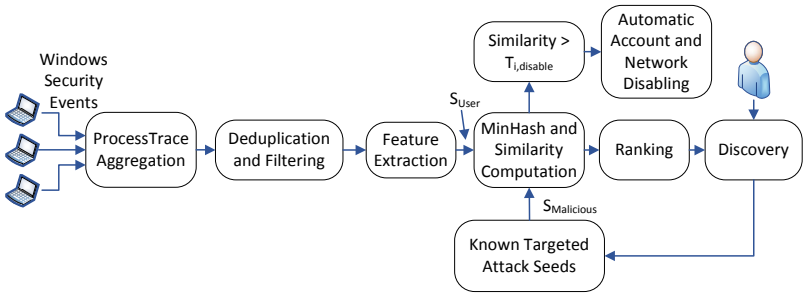

Fig. 1: MART System Overview.

combined to provide the final set of token features.

N-Gram Features: Similar to a rule-based targeted attack detection system, a token feature for a malicious process which exactly matches the entire path of a previous detection is a strong indicator of compromise (IoC). However, attackers may introduce small changes such as renaming the file (e.g. malware7.exe to malware8.exe) which prevents the token-based features from being triggered. To handle these types of obfuscations, we also consider N-Gram features. In MART, the N-Gram features are constructed by a moving window sequence of length N characters for each *ProcessTrace* string in the input collection. For example with $N = 3$, the first four trigram features of C:\temp\malware7.exe are {C:\, :\t, \te, tem}. Most of these trigram features in this example are also shared with C:\temp\malware8.exe. The only ones which differ from the first example are {re7, e7., and 7.e}. As a result, the similarity score between C:\temp\malware7.exe and C:\temp\malware8.exe will still be very high while allowing for some amount of polymorphism by the attacker.

MinHash Filtering: A naive method to compute all similarity scores is to take a cross product of these two collections and compute the similarity score for each pair. This approach is not feasible because it is order $O(UM)$ where U and M are the number of features in S_{User} and $S_{Malicious}$, respectively. Even after filtering, U can be on the order of tens or hundreds of millions of *ProcessTraces* each day, and over time, M can range from thousands to tens of thousands depending on the number of malicious seeds which are being tracked. A linear scan which compares each pair of samples from both sets can easily reach one trillion similarity computations. Instead, we resort to probabilistic techniques to filter the candidate pairs on which we compute our similarity score. In particular, we utilize the MinHash algorithm [9] — an existing randomized hashing technique — to significantly reduce the number of pairs that need to be considered. Each feature in the set (e.g. "net.exe" for tokens or "et." for trigrams) is hashed using a hashing function and the minimum hash value (called the MinHash) is computed over the set. In our system, we use the Jenkins hash, but alternate hashes such as the Murmur 3 can also be used. This process is repeated N times to form the MinHash vector. The complexity of the MinHash algorithm is approximately linear in terms of the number of elements in both sets, $O(N(U + M))$ which is substantially less than $O(UM)$ for linear scan when U and M are large. In order to further increase the computational efficiency, consecutive elements of the MinHash vector are typically grouped together into *bands* where the number of elements in each band is the band size. The values in the bands are then compared instead of individual MinHash vector elements. After using the MinHash algorithm described above to greatly reduce the number of feasible pairs $(S_{Malicious}, S_{User})$, we then compute the exact similarity score which is described next.

Similarity Computation: After feature extraction, the incoming dataset S_{User} consists of a collection of *ProcessTraces* each having the extracted features as its members. Once a *ProcessTrace* is determined to be malicious, it is added to the collection $S_{Malicious}$. Our aim is to compute a similarity score between S_{User} and $S_{Malicious}$ to identify *ProcessTraces* in S_{User} which closely resemble detected targeted attack activity in $S_{Malicious}$. We consider four types of similarity scores, including unweighted Jaccard Index (UJI), weighted Jaccard Index (WJI), unweighted squared match (USM), and weighted squared match (WSM).

Jaccard Index: We first consider both the unweighted and weighted Jaccard index for computing the first type of similarity score. The unweighted Jaccard index is:

$$Score_{UJI}(P_M, P_U) = |P_M \cap P_U| / |P_M \cup P_U| \quad (1)$$

where P_M is an individual, featurized *ProcessTrace* in $S_{Malicious}$ and P_U is a featurized *ProcessTrace* in S_{User}. Alternatively, the weighted Jaccard index is defined as:

$$Score_{WJI}(P_M, P_U) = \frac{\sum_{i \in P_M \cap P_U} \min(w_{P_M}[i], w_{P_U}[i])}{\sum_{i \in P_M \cup P_U} \max(w_{P_M}[i], w_{P_U}[i])}. \quad (2)$$

In (2), $w_{P_M}[i]$ and $w_{P_U}[i]$ are the weights associated with the ith element of the malicious and user *ProcessTraces*, respectively. As in [10], we employ term frequency, inverse document frequency (TFIDF) weighting for the weighted Jaccard index version of our system where $w_P[i] = N_P[i] \log(1.0/N_C[i])$, $N_C[i]$ represents the number of times item i occurs in the corpus of *ProcessTraces*, and $N_P[i]$ is the number of times item i occurs in *ProcessTrace* P. TFIDF weighting is chosen because it emphasizes infrequent patterns that may be associated with targeted attack malware paths,

process names and command line parameter patterns while minimizing the contribution of very frequent patterns such as the trigram feature "C:\".

Weighted, Squared Match: We next propose a new *weighted squared match* similarity score ($Score_{WSM}$) which is more robust to the mimicry and NOOP evasion techniques proposed by Wagner and Soto [8]. Two types of attacks against intrusion detection systems are for the attacker to insert benign commands, to mimic normal activity, or effective NOOPs, to alter the detection algorithm, into the stream of commands entered into the shell. Both the unweighted and weighted forms of the Jaccard Index are susceptible to these forms of evasion because the denominator in (1) and (2) depends upon the set of commands entered by user S_{User}. If the attacker inserts a group of benign commands into the attack sequence, the number of denominator terms increases which lowers both the unweighted and weighted forms of the Jaccard index. To prevent these additional commands from affecting the similarity score, we only consider features in the user *ProcessTrace* (P_U) which are also included in a malicious seed *ProcessTrace* (P_M). In other words, the insertion of additional process creation events, subdirectories, or parameters into the *ProcessTrace* does not affect the weighted squared match similarity score. To achieve this objective, the weighted squared match similarity score is defined as:

$$Score_{WSM}(P_M, P_U) = \sum_{l=0}^{L-1} I[l](w_M[l]/N_{WSM})^2 \quad (3)$$

where L is the number of features in P_M, $w_M[l]$ is the TFIDF weight associated with the lth feature in P_M, and

$$N_{WSM} = \sqrt{\sum_{l=0}^{L-1} w_M^2[l]}. \quad (4)$$

The indicator function $I[l]$ is 1.0 if P_U contains a feature which matches the lth feature in P_M, and 0 otherwise. For example, $I[l] = 0$ when P_U does not contain the lth feature in P_M. N_{WSM} is a normalization term which ensures the final score ranges from 0.0 when no features match to 1.0 when all features match. Without N_{WSM}, MART cannot provide a consistent threshold for automated detection and blocking.

Unweighted, Squared Match: We also propose the unweighted, square match (USM) similarity score, where the $w_{Malicious}[l]$ weights are set to 1.0 in (3) and (4) for all l. The unweighted, squared match score reduces to:

$$Score_{USM}(P_M, P_U) = \sum_{l=0}^{L-1} I[l]/L. \quad (5)$$

Automatic Account and Network Disabling: MART is primarily designed to work in an automated detection and disabling mode. The threshold $T_{i,disable}$ in Figure 1 can be tuned to automatically *disable* network access for the the compromised user account and the compromised computer in the event that the similarity score exceeds this threshold for each individual malicious seed i or a group of seeds. Alternatively, the system administrator can tune a single threshold $T_{disable}$ which is used by all seeds.

Discovery: For automatic account and network disabling, the threshold $T_{i,disable}$ must be set conservatively to avoid too many false positives. MART can be further improved by analysts discovering new, related attacks which have a relatively high similarity score but do not exceed $T_{i,disable}$. There are several methods to aid the discovery process. The first is to generate an *alert* for analysts if the similarity score exceeds a second tuned threshold $T_{i,alert}$ for malicious seed i. The second method is to return the top K nearest neighbor (KNN) results [11] for each *ProcessTrace* in the malicious set. MART supports both methods. The first method is preferable for operational deployment because it only notifies analysts of a possible attack depending on the selected *alert* threshold. The second method can be used by analysts who want to search for potential new attacks which are not highly similar to currently known attacks. The system then takes the feedback from the analysts to improve future detections. Any new detections are added to the set of malicious seeds to detect future intrusions by the same attacker or another attacker who is using similar methods.

Malicious Seed Initialization: Since MART relies on the similarity computation between the incoming user *ProcessTraces* and the set of malicious *ProcessTrace* seeds, the malicious seed set must first be initialized. There are several methods for accomplishing this task. Security companies routinely issue detailed reports on specific instances of targeted attacks. One strategy is to initialize seeds based on the IoCs provided in these reports. Another initialization method is to employ malicious *ProcessTraces* generated by penetration testers as the initial seed set. A third method is for security experts to predict malicious activity related to targeted attacks and data exfiltration and add these behaviors to the initial seed set. Instead of creating their own seed set, an organization might choose to use a cybersecurity service that implements MART to utilize known attacks across a number of customers in order to provide better protection for everyone.

Efficient MapReduce Implementation: To process extremely large datasets, we have implemented MART to run entirely on COSMOS, Microsoft's internal MapReduce platform [12]. The individual components are implemented in a combination of SCOPE [12] and C#. Similar to Hadoop's Hive, SCOPE provides a SQL-like language for efficiently processing large amounts of data. When an operation cannot be computed using SCOPE, we write a custom processor (i.e. mapper) or reducer in C#.

III. DataSets

In this section, we describe the labeled small- and large-scale datasets used for our evaluation of MART.

Small-Scale Dataset1: The organization initially provided researchers with a relatively small dataset including process creation Windows event logs (i.e. Windows security event 4688) from 1,473 computers collected over a two month period. During this time, 1,348 distinct users accessed the

computers and generated 7,849,832 *ProcessTraces*. The computers were selected for close monitoring by analysts for a number of reasons including they were deemed to be high-value assets, had been the target of a previous attack, or had been infected by malware. Often, network analysts will monitor a suspicious computer for an extended length of time in order to understand the targeted attack activity, and the logs included known attacks from two distinct threat actors. All of these computers were equipped with operating systems including Windows 8 or older versions. Thus these computers were able to log the process command names and paths (e.g. C:\Windows\System32\net.exe), but none had the capability of logging process parameters.

Large-Scale Dataset2: The second dataset was provided to the researchers because it also included a known attack and consists of process creation Windows event logs collected from 231,175 computers running on the same network as Dataset1. This large-scale dataset was collected over a single day approximately six months after Dataset1 was collected. For this dataset, 322,539 unique users accessed these computers during the 24-hour period, and the number of *ProcessTraces* that were input to MART was 379,070,572. Some computers on the network were running Windows 8.1 which supports process parameter logging. Thus in addition to the larger scale, the second dataset differs from the first because many of the *ProcessTraces* include the command line parameters in addition to the path and process name.

Targeted Attack Labeled Traces: The labeled targeted threat activity was provided by the organization's network analysts, and the two datasets contained three separate attacks. All of the attacks were detected by specific IoCs associated with known targeted threat actors including connections to their command and control networks as well as other specific patterns employed by these threat actors. Additional indications that the attacks were targeted included the targeting of high value accounts and assets. Once the initial infected computers were identified, individual malicious *ProcessTraces* were then determined by manual inspection.

Data Security and Privacy: This study was approved by our institutional review board. Several steps were taken to protect the security and privacy of the datasets. The access to the raw data was restricted to the researchers. A separate, isolated MapReduce cluster was used to analyze the data and perform the experiments conducted in this study.

IV. Detection of Confirmed Targeted Attacks

In this section, we evaluate the MART algorithm on the two datasets described in Section III. For all experiments, we set the MinHash vector length to 10 and the band size to 1 as discussed in Section II, and we also choose $N = 3$ for the N-Gram (i.e. trigram) features. We calculate the k-nearest neighbors to the dataset's malicious *ProcessTraces* with $k = 25$. For items which are included in two or more lists, we retain only the instance which is the closest (i.e. has the highest similarity score) to one of the seeds and filter the duplicate entries.

Small-Scale Study: In the first experiment we conduct a hindsight study on the small Dataset1. Beginning with 4 known malicious *ProcessTraces* listed in Table II that are initially identified as the attack, we evaluate all eight algorithms, formed by the combinations of tokens versus N-Grams, squared match versus Jaccard index, and unweighted versus weighted, on the two months of data to search for activity from the same targeted attack as well as *ProcessTraces* from other targeted attacks. The "Comp" column specifies the computer identifier and indicates that the attacker visited three computers in this particular attack. The "Month", "Day" and "Time" columns correspond to the relative date and local time where the computers are located. The *ProcessTrace* is aggregated from the attack data. Individual malware files are denoted as $IOC_{x,y}$ where x is the ID of the threat actor and y indicates the malware executable file employed by the actor.

We compare the *ProcessTrace* similarity scores of the 25 K nearest neighbors for the WSM and WJI models with N-Gram features in Figure 2. The weighted squared match similarity model offers a larger value for the highest ranked attack trace allowing easier threshold detection. It also provides a slightly higher spread between the top ranked malicious and benign items compared to the weighted Jaccard index model. Based on these features and its robustness to mimicry attacks, the WSM model outperforms the WJI model on this dataset. The top ranked items in all eight models include the seven *ProcessTraces* from Targeted Attack 1 listed in Table III. The weighted squared match similarity score in the table (column "WSM Sim") is computed using the TFIDF weighted squared match model using N-Gram features.

The MART system is designed to detect similar, but new attacks based on the malicious seeds. From the results in Figure 2, we see that the system performs as expected in that one malicious trace from a second, unrelated threat actor (Targeted Attack 2) is also ranked in the top 6 most suspicious traces. The *ProcessTrace* for this unrelated attack is listed in Table IV and has a WSM Similarity Score = 0.4740. In addition to actual attacks, the system also detects activity from penetration testers labeled as "Red Team". We also include "Security Analyst" items in the figure which demonstrates that their activity is often similar to that of the attackers.

Large-Scale Study: After being able to discover highly ranked attacks in the small-scale dataset, the organization next provided the researchers access to the much larger Dataset2 for analysis. The details of the *ProcessTraces* used in the attack are provided in Table VI. For the training set, we used nine malicious *ProcessTraces*, seven from Targeted Attack 1 from DataSet1 and two from the attack in this dataset which are highlighted in bold in Table VI. The test set contains the remaining 379 million *ProcessTraces* in the dataset. The parent process of the first *ProcessTrace* is Powershell, and the parent processes of all of the remaining entries are Windows command shells. A C# file stored in the infected users' AppData local temp directory is first run. We cannot confirm that this was part of the attack, but the C# file was stored in the same subdirectory as the malware that is later run by the

Comp	Month	Day	Time	*ProcessTrace*
1	2	19	6:22 AM	C:\Windows\SysWOW64\$IOC_{1,1}$;C:\Windows\SysWOW64\PING.EXE;C:\Windows\Temp\<$IOC_{1,2}$>; C:\Windows\SysWOW64\net.exe;C:\Windows\SysWOW64\qwinsta.exe;C:\Windows\SysWOW64\nslookup.exe; C:\Windows\Temp\<$IOC_{1,3}$>;C:\Windows\Temp\<$IOC_{1,4}$>;C:\Windows\SysWOW64\tasklist.exe; C:\Windows\SysWOW64\HOSTNAME.EXE;
2	2	19	9:19 AM	C:\Windows\SysWOW64\ipconfig.exe;C:\Windows\SysWOW64\qwinsta.exe; C:\Windows\SysWOW64\tasklist.exe; C:\Windows\SysWOW64\nslookup.exe; C:\Windows\Temp\<$IOC_{1,3}$>;C:\Windows\Temp\<$IOC_{1,4}$>;
2	2	19	9:32 AM	C:\Windows\SysWOW64\HOSTNAME.EXE;C:\Windows\SysWOW64\qwinsta.exe;C:\Windows\Temp\<$IOC_{1,5}$>; C:\Windows\SysWOW64\$IOC_{1,1}$;C:\Windows\SysWOW64\ipconfig.exe;C:\Windows\SysWOW64\net.exe; C:\Windows\SysWOW64\systeminfo.exe;C:\Windows\SysWOW64\PING.EXE;C:\Windows\Temp\<$IOC_{1,2}$>; C:\Windows\Temp\<$IOC_{1,3}$>;
3	2	19	10:20 AM	C:\Windows\Temp\<$IOC_{1,6}$>;C:\Windows\SysWOW64\$IOC_{1,1}$;C:\Windows\SysWOW64\<$IOC_{1,5}$>; C:\Windows\SysWOW64\ipconfig.exe;C:\Windows\SysWOW64\net.exe;C:\Windows\SysWOW64\tasklist.exe; C:\Windows\SysWOW64\NETSTAT.EXE;C:\Windows\SysWOW64\systeminfo.exe; C:\Windows\SysWOW64\PING.EXE;C:\Windows\Temp\<$IOC_{1,4}$>;

TABLE II: Initial four seeds used to discover other malicious *ProcessTraces* in the Dataset1.

Comp	Month	Day	Time	WSM Sim	*ProcessTrace*
4	1	1	1:58 AM	0.4731	c:\windows\syswow64\qwinsta.exe;c:\windows\syswow64\ping.exe;
4	1	1	2:03 AM	0.86807	c:\windows\syswow64\nslookup.exe;c:\windows\syswow64\ping.exe;c:\windows\syswow64\arp.exe; c:\windows\syswow64\net.exe;c:\windows\syswow64\wbem\$IOC_{1,7}$;c:\windows\syswow64\cmd.exe; c:\windows\syswow64\$IOC_{1,1}$;c:\windows\temp\<$IOC_{1,2}$>;c:\windows\temp\<$IOC_{1,4}$>; c:\windows\temp\$IOC_{1,8}$;
1	1	1	2:57 AM	0.8360	c:\windows\syswow64\<$IOC_{1,9}$>;c:\windows\syswow64\find.exe;c:\windows\syswow64\$IOC_{1,1}$; c:\windows\syswow64\$IOC_{1,8}$;c:\windows\temp\<$IOC_{1,3}$>;c:\windows\syswow64\ipconfig.exe; c:\windows\syswow64\net.exe;c:\windows\syswow64\systeminfo.exe;c:\windows\temp\<$IOC_{1,5}$>; c:\windows\temp\<$IOC_{1,10}$>;
1	1	9	5:41 AM	0.5026	c:\windows\syswow64\ping.exe;c:\windows\syswow64\tasklist.exe;
1	1	9	5:44 AM	0.5138	c:\windows\syswow64\ipconfig.exe;c:\windows\syswow64\ping.exe; c:\windows\syswow64\<$IOC_{1,7}$>;
4	2	20	12:13 PM	0.5030	c:\windows\syswow64\tasklist.exe;c:\windows\syswow64\qwinsta.exe; c:\windows\syswow64\systeminfo.exe;
4	2	20	1:02 PM	0.4537	c:\windows\syswow64\qwinsta.exe;c:\windows\syswow64\tasklist.exe;

TABLE III: Malicious *ProcessTraces* discovered in Dataset1 corresponding to the first targeted attack.

c:\windows\syswow64\net.exe;c:\windows\syswow64\tasklist.exe; c:\windows\syswow64\ping.exe; c:\$IOC_{21}Path\IOC_{21}$; c:\windows\syswow64\hostname.exe;c:\windows\syswow64\sc.exe; c:\windows\syswow64\ipconfig.exe;c:\windows\syswow64\schtasks.exe;

TABLE IV: A *ProcessTrace* from the second targeted attack discovered in Dataset1.

attacker. The actor executes a "net use" and then enters a series of commands running locally stored malware targeting first a computer specified by an IP address and then another computer specified by its hostname. The attacker deletes the remote connections for both target computers, and then immediately appears to repeat the process.

We can make several observations from this attack log. The attack was highly targeted—the log indicates that this session was a return visit by the attacker to this particular computer. The attacker knew the specific IP address of the first destination computer as well as the hostname of the second targeted computer. The attack occurred at night when the majority of the system administrators were sleeping. The second entry indicates that the attack was conducted, in part, by a human, and not a script, given that "use" is misspelled as "ue" and then corrected on the next entry. Finally, the attack is fairly concise with the majority of the activity spanning a period of 46 minutes.

Using this large-scale dataset, we can now evaluate how

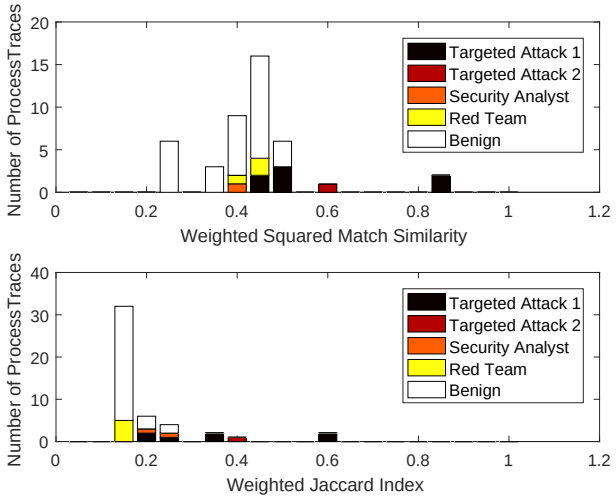

Fig. 2: Comparison of similarity scores for the WSM model ($Score_{WSM}$) (top) and WSI model ($Score_{WJI}$) (bottom) for the Dataset1 N-Gram features.

MART performs based on using the two malicious *ProcessTrace* seeds in the table denoted in bold type plus the seven seeds used from Dataset1. Although the attack was quickly detected and the computer disconnected from the network, these results indicate how well the system might perform if this

Model	Targeted Attack 1
USM Tokens	3
UJI Tokens	3
WSM Tokens	9
WJI Tokens	4
USM N-Grams	3
UJI N-Grams	3
WSM N-Grams	3
WJI N-Grams	5

TABLE V: Model statistics for Dataset2.

particular attacker managed to regain entry on a subsequent day and entered these commands. The weighted squared match similarity score in the "WSM Similarity" column is indicated for items returned in the k-nearest neighbors with $k = 25$. The items with a blank weighted squared match score are not included in the ranked list. We include the seed with "ping.exe -4 -n 1" which is trying to ping a computer using IPV4 using only a single attempt. This parameter combination appears to be unique where the number of network connections is minimized.

The summary of the number of true positive detections is given in Table V, including the unweighted models. These results indicate that the token features can lead to more detections than N-Gram features at large scale assuming the attacker re-used the same commands in subsequent attacks. For example, the TFIDF WSM token-based model detects 6 more attack traces than the WSM N-Gram version. In Figure 3, we compare the KNN results for the WSM and WJI models for the N-Gram features. In general, the WSM similarity score is much closer to 1.0 for the highest ranked attack traces leading to an easier task of tuning a detection threshold. No security analyst activity is detected by any method presumably because the computer was taken off-line so quickly.

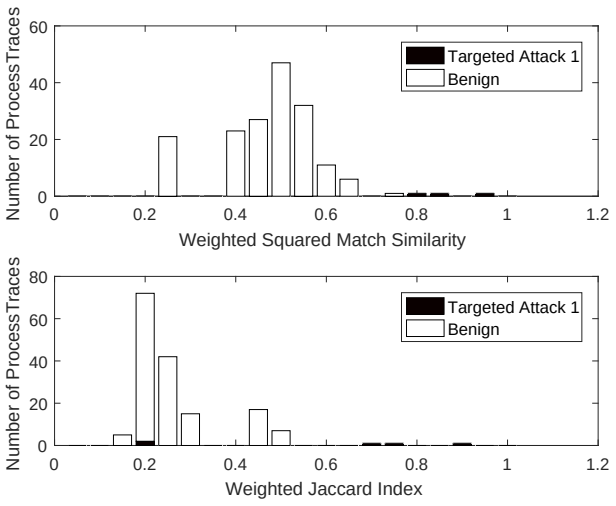

Fig. 3: Comparison of similarity scores for the WSM model ($Score_{WSM}$) (top) and WSI model ($Score_{WJI}$) (bottom) for the Dataset2 N-Gram features.

At large-scale, using TFIDF weighting is important for detecting targeted attacks. The TFIDF weighted systems mostly outperform the unweighted variants in the large-scale analysis in Table V. Two main factors contribute to this result. First the large-scale experiment includes almost 50 times more *ProcessTraces* allowing the rare attack features to have significantly higher weights compared to the small-scale experiment where the attack traces are *relatively* more common. Second the small-scale experiment does not contain any parameters in the process creation logs which significantly reduces the effectiveness of the weighting. Without including the parameters, the only features that are assigned higher weights are malware executable names which are not frequently used in the attacks.

In general, the token features outperform the N-Gram features in the large-scale experiment. This result is to be expected because these particular attackers did not include polymorphism in their activity. As noted previously, the token features yield higher precision. If the attacker re-uses a set of commands or parameters exactly or re-visits a previously infected computer, the token features can produce higher similarity scores. However, we believe that it is more important to be robust to polymorphic behavior. In experiments not presented in this paper, we found that N-Gram features can be used instead of token features without sacrificing too much precision. Therefore we recommend using N-Gram features instead of token features.

Finally the study indicates that the performances of the weighted squared match systems are comparable to their weighted Jaccard index counterparts but offer additional robustness to mimicry and NOOP attacks. The best performing system in Table V is the TFIDF weighted squared match token variant. This system discovers almost twice the number of malicious *ProcessTraces* as the next best system.

Finally, running MART on a MapReduce system is very efficient. For example, executing a highly optimized script for the UJI Token system on Dataset2 requires 1 hour 25 minutes to complete.

V. Evasion

A number of papers have investigated evasion techniques of intrusion detection systems (IDSs) [13], [8]. We next consider possible evasion methods identified by these research efforts related to the proposed MART system.

MART relies on analyzing process command line event logs. If the threat actor is able to employ tactics which avoid generating process creation events, MART will not be able to detect this attack.

Attackers can also attempt to evade detection by reordering the sequence of commands as well as the input parameters for each command they type into the remote shell. One important characteristic of MART is that, depending upon the features, the algorithm is either *completely* invariant or *mostly* invariant to command and parameter reordering. During the first stage of MART, the MinHash algorithm computes the minimum hash value of all token or N-Gram features

Day	Time	WSM Similarity	ProcessTrace
1	11:43 PM		C:\Windows\Microsoft.NET\Framework64\v4.0.30319\csc.exe /noconfig /fullpaths ? ““C:\Users\<infected user>\AppData\Local\Temp\<csharpFile>””; C:\WINDOWS\system32\conhost.exe 0xffffffff;
2	3:50 AM		C:\Windows\SysWOW64\net.exe ue; C:\WINDOWS\system32\conhost.exe 0xffffffff;
2	3:51 AM		C:\Windows\SysWOW64\net.exe use; C:\WINDOWS\system32\conhost.exe 0xffffffff;
2	3:54 AM	1.0	**C:\Users\<infected user>\AppData\Local\Temp\Low\<IOC_{31}> –host <IP Addr>–password <PW>–cmd info; C:\WINDOWS\system32\conhost.exe 0xffffffff;**
2	3:54 AM	1.0	**C:\Windows\SysWOW64\ping.exe -4 -n 1 <HostName>; C:\WINDOWS\system32\conhost.exe 0xffffffff;**
2	3:55 AM	0.9472	C:\Users\<infected user>\AppData\Local\Temp\Low\<IOC_{31}> –host <IP Addr>–password <PW>–ens; C:\WINDOWS\system32\conhost.exe 0xffffffff;
2	3:56 AM	1.0	C:\Users\<infected user>\AppData\Local\Temp\Low\<IOC_{31}> –host <IP Addr>–password <PW>–cmd info; C:\WINDOWS\system32\conhost.exe 0xffffffff;
2	3:56 AM	0.8601	C:\Users\<infected user>\AppData\Local\Temp\Low\<IOC_{31}> –host <HostName>–password <PW>–cmd info; C:\WINDOWS\system32\conhost.exe 0xffffffff;
2	3:58 AM	0.8601	C:\Users\<infected user>\AppData\Local\Temp\Low\<IOC_{31}> –host <HostName>–password <PW>–cmd info; C:\WINDOWS\system32\conhost.exe 0xffffffff;
2	3:58 AM	0.8073	C:\Users\<infected user>\AppData\Local\Temp\Low\<IOC_{31}> –host <HostName>–password <PW>–ens; C:\WINDOWS\system32\conhost.exe 0xffffffff;
2	4:00 AM		C:\Windows\SysWOW64\net.exe use \\<HostName>\<share1>/user:““; C:\WINDOWS\system32\conhost.exe 0xffffffff;
2	4:01 AM	0.8601	C:\Users\<infected user>\AppData\Local\Temp\Low\<IOC_{31}> –host <HostName>–password <PW>–cmd info; C:\WINDOWS\system32\conhost.exe 0xffffffff;
2	4:01 AM		C:\Windows\SysWOW64\net.exe use; C:\WINDOWS\system32\conhost.exe 0xffffffff;
2	4:02 AM		C:\Windows\SysWOW64\net.exe use \\<IP Addr>\<share1>/delete; C:\WINDOWS\system32\conhost.exe 0xffffffff;
2	4:02 AM		C:\Windows\SysWOW64\net.exe use \\<HostName>\<share1>/delete; C:\WINDOWS\system32\conhost.exe 0xffffffff;
2	4:03 AM		C:\Windows\SysWOW64\net.exe use \\<IP Addr>\<share1>; C:\Windows\SysWOW64\net.exe use \\<HostName>\<share1>; C:\WINDOWS\system32\conhost.exe 0xffffffff;
2	4:04 AM	1.0	C:\Users\<infected user>\AppData\Local\Temp\Low\<IOC_{31}> –host <IP Addr>–password <PW>–cmd info; C:\WINDOWS\system32\conhost.exe 0xffffffff;
2	4:04 AM	0.5674	C:\Users\<infected user>\AppData\Local\Temp\Low\<IOC_{31}> –host <HostName>–password <PW>–cmd info; C:\WINDOWS\system32\conhost.exe 0xffffffff;
2	4:05 AM		C:\Windows\SysWOW64\net.exe use \\<IP Addr>\<share1>/delete; C:\Windows\SysWOW64\net.exe use \\<HostName>\<share1>/delete; C:\WINDOWS\system32\conhost.exe 0xffffffff;
2	4:36 AM		C:\Windows\SysWOW64\net.exe user <Expired User Account>/domain; C:\WINDOWS\system32\conhost.exe 0xffffffff;
2	4:36 AM		C:\Windows\SysWOW64\net.exe user <Current User Account>/domain; C:\WINDOWS\system32\conhost.exe 0xffffffff;

TABLE VI: Attack details for Dataset2. The two *ProcessTraces* denoted in **bold** type were included as targeted attack seeds, along with the four malicious seeds in Dataset1 experiment, for this analysis. Items with a blank Jaccard index were not returned in the k-nearest neighbor results with $k = 25$.

which have been extracted from the *ProcessTrace*. For token features, the computation does not depend on the order of the features, and therefore, is not affected by an attacker reordering the commands or parameters. On the other hand, N-Grams produce a slightly different set of features depending on the parameter ordering. Consider the example in Table VII. There are two different trigrams for each of these two sequences: “ 2 ” and “2 -” for the first and “e1 ” and “1 -” for the second. It is possible that these trigrams may be responsible for the minimum hash value for that particular instance of the hash computation. However since we use a band size of 1, it is extremely unlikely that another hash computation will not include the user *ProcessTrace* in the filtered set. In the second stage, the similarity score is also calculated based on the set of features. For token features, the WSM or Jaccard index computed for these two commands is 1.0. For N-Gram features, the similarity score changes by a small amount.

Malicious Seed	New Attack Trace	UJI	USM
ping -n 2 -4 machine1	ping -4 machine1 -n 2	0.810	0.895
ping -n 2 -4 machine1	ping -4 machine1 -n 2 dir c:\windows\system32	0.405	0.895

TABLE VII: *ProcesTrace* parameter reordering for N-Gram features.

For example, the unweighted Jaccard index drops slightly to 0.810 and the unweighted squared match score drops even less to 0.895 for trigrams, although these values will most likely ensure that the unknown *ProcessTrace* remains highly ranked and detectable using either method. MART can be extended to compute the N-Gram features in isolation for each parameter given the command. Doing so would yield an N-Gram-based system which is completely invariant to parameter reordering at the expense of increased complexity. In the case of operating system process command usage, the attacker can also try to vary parameter and process execution orders to avoid detection. However in this case, the MinHash sequence and process parameter invariance described above will help minimize missed detections.

A mimicry attack [8] has been proposed as a method for evading detection by intrusion detections systems. In trojan malware, the attacker embeds malicious code in a benign program. Since execution of the program results in a mostly *normal* set of system calls, an IDS may fail to detect the execution of the malicious code. In a mimicry attack [8], the malicious code simply generates a set of normal system calls directly instead of embedding malicious code into a benign program. A similar attack proposed by Wagner and Soto [8] is the NOOP attack where additional system calls are inserted into the file in order to hide the malicious calls in the "noise" of the useless benign system calls. The WSM similarity score is designed to be robust against mimicry and NOOP attacks because it only considers features which are contained within the attack; inserting additional benign system calls has no effect on the score. Embedding benign commands into the Jaccard index system can reduce the Jaccard index below the threshold allowing the attacker to avoid detection. For example, if the attacker adds the NOOP command *dir c:\windows\system32* to the desired attack in Table VII, the new unweighted Jaccard index drops significantly from 0.895 to 0.405. However, the unweighted squared match score remains unchanged at 0.895. The attacker could attempt to cause false positive detections by entering commands which are highly similar to the malicious seeds without actually performing the true malicious activity. This attack is, however, a dangerous game and will likely not be pursued. An advanced attacker will most likely not want to draw any attention from network analysts to computers which have been successfully infected.

Threat actors can also attempt to use different malware names and locations to avoid detection, but patterns in the operating system command usage can still aid in detection of targeted attacks. The system relies on two broad types of correlation between previously detected attacks and similar attacks occurring in incoming data streams including a) operating system command and parameter usage and b) malicious file names, paths, and parameter usage. For malware executed during a targeted attack, the correlation is reflected in the path, filename, and parameters. For the operating system calls involved in the attacks (e.g. net use), the correlation comes from the command names and parameter usage exhibited during the other attacks. We see from the attack detailed in Table VI, both types of correlations help to provide a high WSM similarity score. In other words, if the attacker changes the malware location and parameters, the malicious *ProcessTrace* may still be detected by the operating system usage as demonstrated by the results on Dataset1. It is important to note that MART is designed to operate in part by analyzing log files. Even if an attacker does change the malware names but still follows some pattern, MART can be run in a hindsight mode to detect past intrusions if the organization has saved their logs for some period of time — the attackers cannot undo their past activity assuming they cannot overwrite the older log files.

VI. Related Work

Very few papers have been published on the analysis and detection of targeted attack activity. Balduzzi et al. [1] propose a system to cluster the hostnames and requests from URLs and individual computers based on telemetry from Trend Microsystem's anti-virus engine. Next, they further group these two types of URL clusters and machines for client machines in related industries such as oil and gas or banking. By evaluating these URL clusters and machines within a particular group, they are able to detect potential targeted attack activity. Mandiant published a white paper detailing cyber esponiage activity by a group they call APT1 [14].

An early survey of intrusion detection systems was written by Brown et al. [15]. Anand [16] provides a more recent overview on intrusion detection. The most popular network intrusion detection systems include Bro [17] and Snort [18]. Two important early papers on applying the RIPPER rule-learning algorithm to intrusion detection include Lee et al. [19] and Lee et al. [20]. Two highly cited intrusion detection systems for unix systems include stide [21] and pH [22]. Portnoy et al. [23] investigated clustering in the context of intrusion detection. A well known paper on event correlation for intrusion detection is by Debar and Wespi [24].

MART falls into the general category of misuse host intrusion detection systems. Our system is most closely related to that of Liao and Vermuri [10] which uses K-nearest neighbor clustering and TFIDF weighting for host intrusion detection. However, their system uses simple linear scan and does not utilize the MinHash algorithm for fast computation. It also analyzes system calls from applications instead of process calls as we do in MART. In addition, MART uses the new weighted, squared match similarity score. They analyze the 1998 DARPA Basic Security Model (BSM) dataset with *simulated* attacks which is known to not be representative of real-world attacks. Our system analyzes data with actual targeted intrusion activity on a large-scale organizational network. Furthermore, they look for similarity among all normal programs while MART detects similarity to known attacks. Finally, their system only records the name of the system calls and does not include paths or parameters. In addition to the process names, our small-scale dataset includes the paths and both paths and parameters were logged in many cases in our large-scale dataset.

The unweighted Jaccard index was previously used to cluster malware files in [5], [7]. Brode first introduced the

MinHash algorithm in 1997 [25] and considered it in more depth in [26]. A good overview of the MinHash algorithm is provided by Rajaraman and Ullman [9]. A weighted MinHash algorithm has been proposed by Ioffe [27]. In the context of security applications, the MinHash algorithm was previously used to estimate the unweighted Jaccard index and investigate account hijacking on twitter [28]. Chum et al. [29] investigate the use of the MinHash algorithm and TFIDF weighting for vision tasks.

VII. Conclusions

Finding intrusions in large computer networks is a daunting task in terms of the number of analysts needed to investigate potential detections as well as the computational power required to correlate activity produced by a resilient attacker who may leave many backdoors in the network. To help address these important problems we have developed MART — a distributed analysis tool that enables investigators to find malicious human activity on a large computer network monitored with event logging enabled on each host. The results on the two targeted attack datasets are highly encouraging. Targeted attack activity is included at the top of the ranked results in both cases. The system is designed to discover additional activity from the same actor as well as similar activity from unrelated actors; the inclusion of the second targeted attack in the results for Dataset1 provides some evidence that detecting new threats is indeed possible. Although previous research argues that detection of rare intrusion events is extremely problematic [30], the Dataset2 results indicate that MART can successfully operate at very large scale. The new weighted squared match similarity score provides robustness to mimicry and NOOP attacks. While not being the top performing system for either dataset, we believe the weighted squared match N-Gram version of MART offers the best combination of performance and resiliency to evasion techniques. Ultimately, we envision MART as another layer in an organization's defense in depth strategy.

References

[1] M. Balduzzi, V. Ciangaglini, and R. McArdle, "Targeted attacks detection with spunge," *Proceeding of Conference on Privacy, Security and Trust (PST)*, pp. 185–194, 2013.

[2] K. Krombholz, H. Hobel, M. Huber, and E. Weippl, "Social engineering attacks on the knowledge worker," *Proceedings of the 6th International Conference on Security of Information and Networks (SIN)*, pp. 28–35, November 2013.

[3] Microsoft, "Mitigating pass-the-hash and other credential theft, version 2," http://www.microsoft.com/en-us/download/details.aspx?id=36036, 2014.

[4] M. Thomlinson, "Recent cyberattacks," http://blogs.technet.com/b/msrc/archive/2013/02/22/recent-cyberattacks.aspx, 2013.

[5] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, "Scalable, behavior-based malware clustering," in *Proceedings Annual Network and Distributed System Security Symposium (NDSS)*,, 2009.

[6] L. Invernizzi, P. M. Comparetti, S. Benvenuti, C. Kruegel, M. Cova, and M. Cova, "EvilSeed: A Guided Approach to Finding Malicious Web Pages," *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pp. 1265–1276, 2012.

[7] J. Jang, D. Brumley, and S. Venkataraman, "Bitshred: Feature hashing malware for scalable triage and semantic analysis," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2011.

[8] D. Wagner and P. Soto, "Mimicry Attacks on HostBased Intrusion Detection System," *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 255–264, 2002.

[9] A. Rajaraman and J. D. Ullman, *Mining of Massive Datasets*, 2nd ed. Cambridge: Cambridge University Press, 2014.

[10] Y. Liao and V. R. Vemuri, "Using k-nearest neighbor classifier for intrusion detection," *Computers & Security*, vol. 21, no. 5, pp. 439–448, 2002.

[11] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *Proceedings of the IEEE Transactions on Information Theory*, pp. 21–27, 1967.

[12] R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou, "SCOPE: easy and efficient parallel processing of massive data sets," *Proceedings of the Vldb Endowment*, vol. 1, pp. 1265–1276, 2008.

[13] K. M. C. Tan, K. S. Killourhy, and R. A. Maxion, "Undermining an Anomaly-Based Intrusion Detection System Using Common Exploits," *Proceedings of Recent Advances in Intrusion Detection (RAID)*, pp. 54–73, 2002.

[14] Mandiant, "Apt1 exposing one of china's cyber espionage units," http://intelreport.mandiant.com/Mandiant_APT1_Report.pdf.

[15] D. J. Brown, B. Suckow, and T. Wang, "A survey of intrusion detection systems," http://cseweb.ucsd.edu/classes/fa01/cse221/projects/group10.pdf, 2002.

[16] V. Anand, "Intrusion Detection: Tools, Techniques and Strategies," *Proceedings of the Vldb Endowment*, pp. 69–73, 2014.

[17] V. Paxson, "Bro: A System for Detecting Network Intruders in Real-Time," *Computer Networks*, vol. 2, pp. 2435–2463, 1999.

[18] M. Roesch, "Snort - Lightweight Intrusion Detection for Networks," *Proceedings of USENIX Systems Administration Conference (LISA)*, 1999.

[19] W. Lee, S. J. Stolfo, and P. K. Chan, "Learning Patterns from Unix Process Execution Traces for Intrusion Detection," *Proceedings of AAAI Workshop on AI Methods in Fraud and Risk Management*, pp. 50–56, 1999.

[20] W. Lee, S. J. Stolfo, and K. W. Mok, "A Data Mining Framework for Building Intrusion Detection Models," *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pp. 120 – 132, 1999.

[21] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for unix processes," *Proceedings of the IEEE Symposium on Security and Privacy*, p. 120âĂŞ128, 1996.

[22] A. Somayaji and S. Forrest, "Automated response using system-call delays," *Proceedings of the Usenix Security Symposium*, 2000.

[23] L. Portnoy, E. Eskin, and S. Stolfo, "Intrusion detection with unlabeled data using clustering," *Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA)*, pp. 85–103, 2001.

[24] H. Debar and A. Wespi, "Aggregation and Correlation of Intrusion-Detection Alerts," *Proceedings of Recent Advances in Intrusion Detection (RAID)*, pp. 85–103, 2001.

[25] A. Z. Brode, "On the resemblance and containment of documents," in *Proceedings Compression and Complexity of Sequences*, 1997, pp. 21–29.

[26] A. Z. Brode, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations," in *Proceedings Journal of Computer and System Sciences*, 2000, pp. 630–659.

[27] S. Ioffe, "Improved consistent sampling, weighted minhash and l1 sketching," *Proceeding of the IEEE International Conference on Data Mining (ICDM)*, pp. 246–255, 2010.

[28] K. Thomas, F. Li, C. Grier, and V. Paxson, "Consequences of connectivity: Characterizing account hijacking on twitter," *Proceedings of the 21st Conference on Computer and Communications Security (CCS)*, pp. 489–500, November 2014.

[29] O. Chum, J. Philbin, and A. Zisserman, "Near duplicate image detection: min-hash and tf-idf weighting," in *British Machine Vision Conference*, 2008.

[30] S. Axelsson, "The base-rate fallacy and the difficulty of intrusion detection," *ACM Transactions on Information and System Security (TISSEC)*, pp. 186–205, 2000.