System Call-based Detection of Malicious Processes

Raymond Canzanese* and Spiros Mancoridis†
*Dept. of Electrical and Computer Engineering
†College of Computing and Informatics
Drexel University, Philadelphia, PA, USA
{rcanzanese,mancors}@drexel.edu

Moshe Kam
Newark College of Engineering
New Jersey Institute of Technology
Newark, NJ, USA
kam@njit.edu

***Abstract*—System call analysis is a behavioral malware detection technique that is popular due to its promising detection results and ease of implementation. This study describes a system that uses system call analysis to detect malware that evade traditional defenses. The system monitors executing processes to identify compromised hosts in production environments. Experimental results compare the effectiveness of multiple feature extraction strategies and detectors based on their detection accuracy at low false positive rates. Logistic regression and support vector machines consistently outperform log-likelihood ratio and signature detectors as processing and detection methods. A feature selection study indicates that a relatively small set of system call 3-grams provide detection accuracy comparable to that of more complex models. A case study indicates that the detection system performs well against a variety of malware samples, benign workloads, and host configurations.**

I. Introduction

Behavioral analysis techniques use characteristics of executing software to identify potential malware [1]. One such technique is system call analysis, wherein malicious behaviors are identified by their system call traces [2]. System calls are requests for operating system services, such as memory and filesystem access. This study describes a malware detection system that uses system call analysis to detect malware that evade traditional defenses. Such malware include zero-day malware and malware variants. The system is intended to supplement existing defenses such as antivirus software by identifying compromised hosts. The specific configuration of the system is determined through experimental evaluation of multiple feature extraction and detection strategies. Only system call analysis techniques that are 'lightweight' enough to be used for real-time detection in production environments are considered. This study evaluates the detection performance of the system against system call traces collected from recently discovered malware and benign software executed in production environments. More than $55{,}000$ malware samples and 3.5 million system call traces collected from hosts running Microsoft Windows 7, 8, Server 2008, and Server 2012 are used for the experimental evaluation.

The main contribution of this study is the evaluation and comparison of detection and feature extraction techniques at a low false positive rate (FPR) of 10^{-5}. This study demonstrates that logistic regression (LR) and support vector machines (SVMs) trained using stochastic gradient descent (SGD) provide higher detection accuracy than signature and log-likelihood ratio test (LLRT) detectors. A logarithmic bag-of-3-grams feature extraction strategy provides higher detection accuracy than considered alternatives. The detection system generalizes well to detect previously unseen malware samples under various host configurations and benign workloads. A feature selection study is also presented, wherein the most informative system call patterns are identified.

II. Related Work

Behavioral malware detectors are used to address the shortcomings of traditional detectors, especially those based solely on static analysis. Static analysis techniques have enjoyed limited success due to the relative ease with which static features can be obfuscated [3]. Behavioral techniques include audit and performance monitor data analysis, taint analysis, semantic analysis, and control flow graph analysis [4]–[11]. Malware detection using system call analysis has been approached using statistical and signature detectors and feature extraction techniques from document classification [12]–[17].

This study seeks to address two perceived shortcomings in the related work. First, this study seeks to address conflicting claims regarding the effectiveness of feature extraction and detection techniques. It compares multiple feature extraction strategies and detection techniques against a single dataset. Second, this study seeks to characterize detection accuracy at very low FPRs against realistic datasets. To be practically useful, a malware detection system must exhibit high-enough detection accuracy at a low FPR [18]. For the majority of the related work, few claims regarding detection accuracy at low FPRs or the general applicability of the proposed techniques can be made. This is due to the small sample sizes considered for evaluation and the use of specialized sandbox environments for malware analysis [6]. Two studies include extensive experimental evaluation of signature-based detectors, characterizing detection accuracy at a FPR of 10^{-2} [19], [20]. This study characterizes detection accuracy at a FPR of 10^{-5} against system call traces collected from production hosts.

III. System Call Traces

The system call traces used for this study were collected from production hosts using a custom host-agent known as the System Call Service (SCS) [1]. The SCS is a service application

[1]SCS source code: https://github.com/rcanzanese/SystemCallService

```
NtQueryPerformanceCounter
NtProtectVirtualMemory
NtProtectVirtualMemory
NtQueryInformationProcess
NtProtectVirtualMemory
NtQueryInformationProcess
NtQueryInformationProcess
NtQueryInformationProcess
NtQuerySystemInformation
NtQuerySystemInformation
```

Fig. 1. Example system call trace of process (truncated to 10 calls)

TABLE I
2-GRAM, ORDERED (O) AND UNORDERED (U) BAG-OF-2-GRAMS REPRESENTATION OF TRACE IN FIG. 1

n-gram	O	U
NtQueryPerformanceCounter, NtProtectVirtualMemory	1	1
NtProtectVirtualMemory, NtProtectVirtualMemory	1	1
NtProtectVirtualMemory, NtQueryInformationProcess	2	3
NtQueryInformationProcess, NtProtectVirtualMemory	1	–
NtQueryInformationProcess, NtQueryInformationProcess	2	2
NtQueryInformationProcess, NtQuerySystemInformation	1	1
NtQuerySystemInformation, NtQuerySystemInformation	1	1

that logs process-level system call traces in recent versions of Microsoft Windows. Fig. 1 provides an example trace, showing the system calls made by a process in the order in which they occurred. For this study, the SCS recorded up to the first $10{,}000$ calls made by each process. Only traces containing at least $1{,}500$ system calls were used in this study to ensure fair comparison of detection results at trace lengths up to $1{,}500$.

A. Feature extraction from system call traces

This study characterizes system call traces using a bag-of-n-grams model. In document classification, a bag-of-words model is the representation of a text document as a vector of word frequencies. Analogously, a bag-of-n-grams model is the representation of a system call trace as a vector of system call n-gram frequencies. An n-gram is a length n sequence of system calls occurring contiguously in a trace, extracted using a sliding window.

Table I shows two bag-of-2-grams representations of the system call trace in Fig. 1. The ordered 2-gram representation considers the local ordering of the calls within the sliding window, whereas the unordered 2-gram representation ignores the local ordering. The '–' symbol indicates that a sequence is a reordering of another, and therefore not considered in the unordered representation. This study compares the detection accuracy afforded by both representations.

Each system call trace can be represented as a length m column vector $\mathbf{x}$ containing the frequency of every possible n-gram. The SCS monitors access to 465 distinct system calls, making $m = O(465^n)$. For a system call trace of length l, where $l \ll m$, $\mathbf{x}$ is sparse. In practice, the vectors are stored in a sparse array format with $O(l)$ storage complexity, and analysis techniques are selected to take advantage of this sparsity.

B. Feature scaling

The system call n-gram frequencies used in this study scale to different orders of magnitude. The feature scaling technique term frequency – inverse document frequency (TF-IDF) transformation is used to account for these differences in scale. TF-IDF transformation is an information retrieval technique used in document classification and intrusion detection [13]. The TF-IDF transformation of a vector $\mathbf{x}$ is the element-wise product of the term frequency (TF), computed directly from $\mathbf{x}$, and the inverse document frequency (IDF), computed from the set $\mathcal{X}$ of all training vectors,

$$\text{TF-IDF}(\mathbf{x}, \mathcal{X}) = \text{TF}(\mathbf{x}) \odot \text{IDF}(\mathcal{X})\,. \tag{1}$$

The TF can be the raw frequency or the logarithmic frequency, wherein Laplace smoothing is used to account for the sparsity of $\mathbf{x}$ [21],

$$\text{TF}(\mathbf{x}) = \mathbf{x} \quad \text{or} \quad \text{TF}(\mathbf{x}) = \log(\mathbf{x} + 1)\,. \tag{2}$$

The IDF applies weights to features inversely with their frequency of occurrence, placing higher emphasis on rare n-grams. Considering p to be the number of traces in $\mathcal{X}$ and $\mathbf{d}$ to be a vector counting the traces in which each n-gram appears, the IDF is

$$\text{IDF}(\mathcal{X}) = \log\left(\frac{1+p}{1+\mathbf{d}}\right) + 1\,. \tag{3}$$

Laplace smoothing is used to account for the sparsity of the data, and the result is offset by one to ensure that every n-gram receives a non-zero weight. After TF-IDF transformation, the feature vectors are scaled to unit magnitude using either the L_1 or L_2 norm. This final step is used to facilitate analysis using the described detection algorithms.

IV. Detection Algorithms

The detection algorithms in this study were selected because they work well for large-scale learning problems with sparse, high-dimensional data and many training instances. The algorithms exhibit low computational complexity during detection and are effective in both document classification and intrusion detection. They are used in a supervised learning context, wherein labeled training data are used for model creation. Each technique includes a training and detection algorithm.

A. Signature-based detector (SIG)

The signature-based detector is desirable due to the simplicity of its training and detection algorithms and its white-box model. The detector compares the system call trace of a process to a set of n-grams observed only in malware processes during training, *i.e.*, the signature set. The signature set is represented as a binary vector $\mathbf{s}$ that indicates which n-grams belong to the set. During detection, a process is labeled 'malicious' if the number of signatures matched by a process exceeds a threshold,

$$\mathbf{s}^T\mathbf{x} > \Lambda\,. \tag{4}$$

B. Multinomial log-likelihood ratio test (LLRT)

The naïve multinomial LLRT is a statistical test used in document classification [22]. It is desirable due to the simplicity of its training and detection algorithms. It models the distributions of the feature data using a categorical distribution model and assumes conditional independence of the features. The probability distributions $p_{x|M}(x)$ for the malware data and $p_{x|B}(x)$ for the benign data are estimated from the training data for each feature $x \in \mathbf{x}$. During detection, a process is labeled 'malicious' if

$$\sum_{x \in \mathbf{x}} \log \left(\frac{p_{x|M}(x)}{p_{x|B}(x)} \right) > \Lambda. \tag{5}$$

The threshold Λ can be chosen under the Bayes criterion, based on error costs and prior probabilities; or under the Neyman-Pearson criterion, by maximizing the true positive rate (TPR) under a false positive rate (FPR) constraint [23]. This study uses the latter approach, since the prior probabilities of malware infection are not known *a priori*.

C. Linear support vector machines (SVMs)

SVMs are advantageous because they do not operate on the same restrictive assumptions as the LLRT regarding the distribution and independence of the data. Linear SVMs seek a hyperplane $\mathbf{w}^T\mathbf{x}$ that optimally separates data points of two classes. In this study, the parameters $\mathbf{w}$ defining the hyperplane are learned from labeled training data using SGD. SGD is an optimization algorithm well suited to large-scale learning problems such as this [24]. The SGD objective function is given in terms of the feature vectors $\mathbf{x}_i \in \mathcal{X}$ and their corresponding labels $y_i \in \{-1, 1\}$. The objective considers a regularization constant α that penalizes high model complexity and loss function L. For soft margin SVMs, L is the hinge loss,

$$L(t, y) = \max(0, 1 - ty)\,, \tag{6}$$

and the objective function is

$$E(\mathbf{w}) = \frac{1}{p} \sum_{i=1}^{p} L(y_i, \mathbf{w}^T\mathbf{x}_i) + \alpha ||\mathbf{w}||_2\,. \tag{7}$$

During detection, a process is labeled 'malicious' if

$$\mathbf{w}^T\mathbf{x} > \Lambda\,. \tag{8}$$

D. Logistic regression (LR)

LR is desirable because its probability estimates can be used with sequential detection or data fusion techniques that require probabilities of observations as input. LR provides a model for estimating the probability that a system call trace comes from a malicious process [25]. The objective function for training the LR detector (7) uses logarithmic loss,

$$L(t, y) = \log\left(1 + \exp\left(-ty\right)\right). \tag{9}$$

During detection, the probabilities that a process is malicious or benign are

$$p_{M|\mathbf{x}}(\mathbf{x}) = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x})}} \tag{10}$$

$$p_{B|\mathbf{x}}(\mathbf{x}) = 1 - p_{M|\mathbf{x}}(\mathbf{x})\,, \tag{11}$$

and a process is labeled 'malicious' if

$$\log\left(\frac{p_{M|\mathbf{x}}(\mathbf{x})}{p_{B|\mathbf{x}}(\mathbf{x})}\right) > \Lambda\,. \tag{12}$$

V. Feature Selection

Feature selection is the process of choosing a subset of the available features to use for detection. The motivation for feature selection is two-fold. First, the storage and processing of high-dimensional datasets introduces memory and processing overhead. Second, including non-informative features during the training can lead to overfitting. This study uses recursive feature elimination (RFE) for feature selection [26]. RFE was chosen because it selects the feature subset that provides the highest detection accuracy. Furthermore, it works with detection algorithms that provide feature weights. Thus, RFE can be used with the LR and SVM detectors. RFE recursively eliminates features from consideration, re-training the detector with the remaining feature subset. The eliminated features are those with the lowest weights computed during training.

VI. Experimental Setup

The described detectors and feature extraction strategies were evaluated against more than 3.5 million system call traces collected by the SCS. The traces were collected from 19 hosts in home and office settings, 15 hosts in public computer laboratories, and 21 hosts in an isolated testbed. The traces were collected between October 2014 and June 2015.

The testbed was designed to provide a production-like environment for the execution of malware. The testbed provided a variety of host configurations (*e.g.*, different OS versions, installed software, security patches) to maximize the probability that the injected malware would perform their malicous tasks. The hosts were connected to an isolated network routing traffic to a Stratagem[2] honeypot. The honeypot was used to mimic a vulnerable host, enabling injected malware to perform network communication. Benign software programs, including system utilities, games, and desktop publishing software, were also executed on the testbed to ensure that the detection results were not biased by the testbed environments.

Ground truth labels were created by cross-referencing checksums of the executable image of each process against the VirusTotal database. Executable images not included in the VirusTotal database were labeled according to their provenance. Those that appeared on the testbed after the execution of a malware sample were labeled 'malicious'. Otherwise, they were labeled 'benign'.

The malware used in this study were collected from publicly available malware archives, blacklisted URLs, and honeypots.

[2]Stratagem, http://sourceforge.net/projects/stratagem

TABLE II
MOST COMMON MALWARE CATEGORIES AND FAMILIES USED IN ANALYSIS

category	count	family	count
Trojan	16,748	Trojan.Generic	8,071
Backdoor	6,744	Trojan-Spy.Zbot	1,536
Virus	4,936	Virus.Sality	1,430
Trojan-Dropper	2,478	Backdoor.DarkKomet	1,205
Worm	2,020	Email-Worm.Mydoom	1,135
Trojan-Spy	1,952	Trojan.Inject	1,101
Trojan-Downloader	1,462	Trojan.Agent	1,020
Email-Worm	1,336	Backdoor.Hupigon	980
Trojan-PSW	1,042	Virus.Parite	794
Trojan-Ransom	953	Virus.Delf	759

The malware were standalone executable files first submitted to VirusTotal between January 2012 and June 2015 and identified as malware by at least 15 antivirus vendors. More than 55,000 malware samples were executed on the testbed, with each sample being executed exactly once. The most common malware categories and families included in the study (according to their Kaspersky[3] labels) are presented in Table II. The malware samples covered 1,057 distinct malware families and 36 distinct malware categories.

VII. Experimental Results

The experimental results were obtained through a process of successive refinement of detector and feature extractor parameters. The experimental results are presented under the Neyman-Pearson criterion in terms of the maximum detector TPR at a fixed FPR of 10^{-5}. The FPR is the fraction of benign processes incorrectly identified as malware. The TPR is the fraction of correctly identified malware samples. A malware sample is considered correctly identified if at least one of its associated processes is positively identified as malware. This approach is used to prevent malware samples that spawn multiple processes from biasing the detection results.

A. Detector comparison

Fig. 2 compares the detection accuracy achieved by each of the detectors described in Section IV. The plots show the TPR of each detector at a fixed FPR of 10^{-5}. Each plot shows the detection results for a fixed n-gram length n while varying the trace length l. The results were achieved using 10-fold cross-validation. Since each malware sample was executed only once on the testbed, these results represent the achievable detection accuracy against previously unseen malware samples. These results were obtained using ordered n-grams, logarithmic TF-IDF transformation, and the L_2 norm.

Fig. 2 shows comparatively poor performance for all four detectors when $n = 1$, indicating that system call frequency alone is insufficient for detection. The highest TPR achieved by a 1-gram detector at an FPR of 10^{-5} did not exceed 30%. For $n \in \{2, 3, 4\}$, the LR and SVM detectors offer significantly improved detection performance over the SIG and LLRT detectors. The comparatively poor performance of the SIG detector is caused by overfitting. The signatures

[3]Kaspersky Lab, http://www.kaspersky.com/

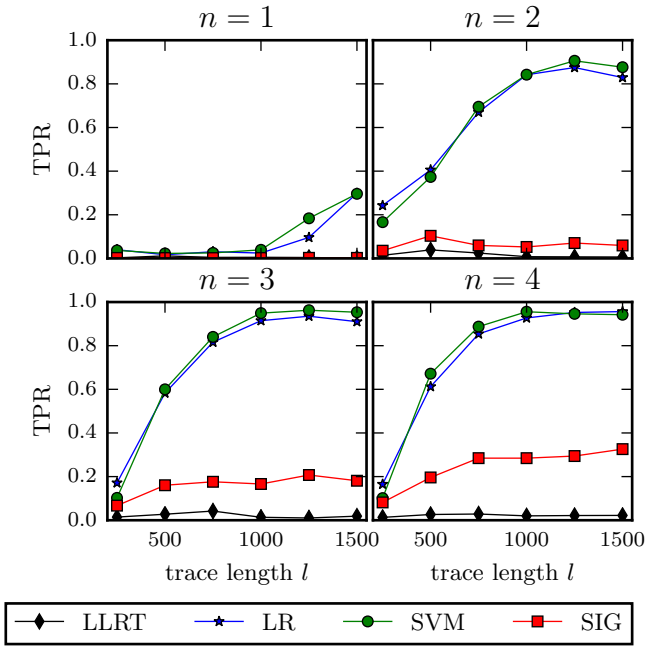

Fig. 2. Maximum TPR at FPR=10^{-5} at sequence lengths $n \in \{1, 2, 3, 4\}$ and system call trace lengths $l \in [250, 1500]$

identified during training are mainly artifacts of specific malware samples that do not generalize to detecting other malware samples. The comparatively poor performance of the LLRT detector is attributed to both overfitting and poor model assumptions, particularly the assumption of conditional feature independence. The TPR of the LR and SVM detectors peaks around 1000 system calls for $n \in \{3, 4\}$, indicating that the studied malware samples were detected during their early stages of execution. The observed behaviors of the malware samples during their initial execution included

- propagating within a host or over the network,
- communicating with a remote server,
- modifying host configurations to hide the malware,
- installing the malware to ensure their permanence,
- spawning additional malware processes,
- collecting information about infected hosts,
- installing additional malware samples, and
- disabling security software and other functions used for malware removal or detection.

Fig. 3 presents the maximum TPR achieved by each detector in Fig. 2 at a fixed FPR of 10^{-5}. The TPR of each detector is shown for n-gram lengths $n \in \{1, 2, 3, 4\}$. The results indicate that detection accuracy generally improves as n increases. For the SVM and LR detectors, increasing n beyond 3 does not provide statistically significant improvement. This is likely the result of overfitting, wherein the benefit of the introduction of additional informative features is offset by those that are not informative. Furthermore, since the number of features scales exponentially in n, using the lowest n value that provides sufficient detection accuracy is desired to minimize system

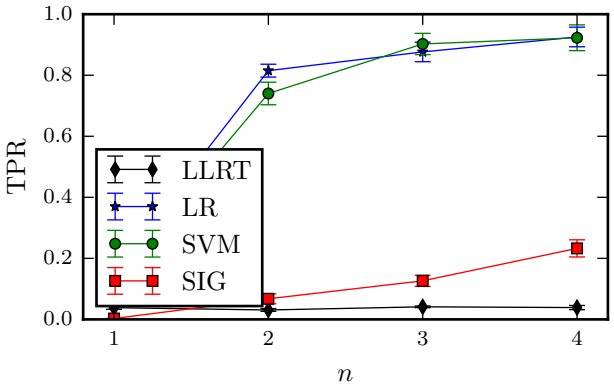

Fig. 3. Maximum TPR observed at FPR $= 10^{-5}$ for $n \in \{1, 2, 3, 4\}$

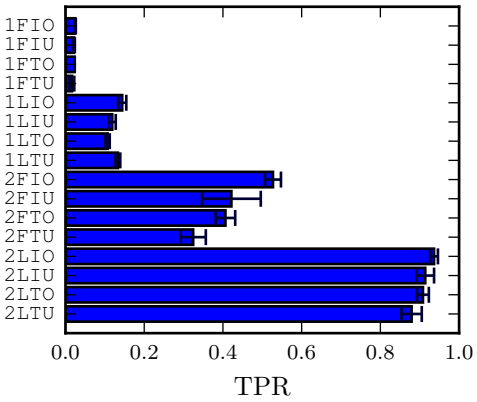

Fig. 4. Maximum TPR achieved at FPR=10^{-5} for the feature extraction strategies described in Section VII-B

overhead. The remainder of this paper focuses solely on the LR detector where n=3.

B. Feature extraction

Fig. 4 shows the effects of various feature extraction strategies on the TPR of the LR detector at a FPR of 10^{-5} using 3-grams of system calls. It compares the detection accuracy afforded by the 16 combinations of the following 4 parameters,

- (`1`) L_1 or (`2`) L_2 norm,
- (`F`) raw TF or (`L`) logarithmic TF,
- (`T`) TF alone or (`I`) TF-IDF transformation, and
- (`O`) ordered or (`U`) unordered n-grams.

The results were achieved through 10-fold cross validation. Fig. 4 shows that the L_2 norm provides higher detection accuracy than the L_1 norm. The L_1 norm was considered because of its straightforward interpretation as the relative frequency of each 3-gram. The comparatively poor performance of the L_1 norm is caused by the SGD training algorithm, which performs better with L_2 normalized features. Fig. 4 also shows that logarithmic frequencies provide better detection performance than raw frequencies. This is because the logarithmic frequencies account for the differing scales of the feature data and place higher emphasis on rare n-grams. IDF transformation also emphasizes rare n-grams, but the effect of using IDF alone is less pronounced. Both of these conclusions are consistent with those from studies of document classification [27].

The bottom four entries in Fig. 4 illustrate the effects of IDF transformation and system call ordering. Ordered 3-grams and IDF transformation outperform their counterparts, and using both provides a significant increase in performance over using TF transformation and unordered 3-grams. As a result, the remaining results consider only the `2LIO` feature extraction configuration.

C. Feature selection

RFE was performed using the LR detector and the `2LIO` feature extraction configuration to identify the subset of ordered 3-grams that provided the highest TPR at fixed FPR of 10^{-5}. The subset that provided the highest TPR contained 3,500 system call 3-grams. Of the 465 system calls monitored by the SCS, 244 calls were present in the chosen feature subset. The most informative 3-grams in the chosen subset included system calls from the following categories. For each category, two of the selected calls are listed (with 'Nt' prefixes omitted for brevity).

- **Files:** LockFile, QueryVolumeInformationFile,
- **LPC:** CreatePort, RequestWaitReplyPort,
- **Memory:** AllocateVirtualMemory, OpenSection,
- **Misc.:** QuerySystemInformation, AllocateUuids,
- **Processes:** QueryInformationProcess, TerminateProcess,
- **Registry:** LockRegistryKey, SetInformationKey,
- **Security:** ImpersonateThread, AdjustGroupsToken,
- **Synchronization:** CreateKeyedEvent, TraceEvent.

The selected feature set contained system calls from every possible category. This result indicates that there are no simple, intuitive rules for selecting informative n-grams. Some approaches in the related work have considered only system call categories that seem intuitively informative, such as filesystem and registry calls. However, this study indicates that all categories of system calls are informative and that feature selection should be performed empirically.

System call diversity is also evident in the selected 3-grams. For example, the selected 3-gram ⟨TerminateThread, CloseObjectAuditAlarm, EnumerateValueKey⟩ covers 3 system call categories, including thread, security, and registry operations. Other selected 3-grams were more homogeneous, including ⟨CreateFile,CreateFile,QueryVolumeInformationFile⟩ and a sequence of three successive calls to EnumerateValueKey.

D. Case study

The preceding sections characterized the achievable performance of the described detection system using 10-fold cross-validation. This section instead uses a leave-one-out cross-validation strategy, wherein the models are trained using all but one of the hosts and tested against the remaining host. This strategy tests how well the detection models generalize to new host configurations, benign workloads, and malware

samples. Here, the models are trained against a subset of the training data and the thresholds are selected against a disjoint subset. The thresholds are chosen to achieve a target FPR of 10^{-5}. The resulting models and thresholds are tested against the system call traces collected from the remaining host.

The leave-one-out cross-validation results provided an overall FPR of 3.2×10^{-5} and a TPR of 0.92. The false positives occurred on 13 of the 55 hosts and resulted from 22 applications. The highest represented applications in this set were hardware vendor monitoring and update applications, installed on the hosts as bundled software. Other applications included a secure erasure utility, a third-party task manager, and 6 applications of unknown provenance that were executed from temporary directories. The missed detections included suspected adware, spyware, Trojans, and backdoors. Among these were DarkMoon, Meredrop, Lamechi, SpyBot, and Yuner.

VIII. Discussions and Conclusions

The goal of this study was to design a behavioral malware detection system that uses system call analysis to detect the execution of malicious processes. The proposed system is intended to supplement existing defenses by identifying compromised hosts, particularly hosts infected with zero day malware and malware variants that evade existing defenses. The specific configuration of the system was determined through experimental evaluation of multiple feature extraction and detection strategies against system call traces collected from production hosts. For feature extraction, the system uses $3{,}500$ ordered system call 3-grams that are TF-IDF transformed using logarithmic frequency and scaled using the L_2 norm. For detection, the system uses a LR detector trained using SGD.

The case study indicated that the detector was able to detect previously unseen malware samples in new environments and under a variety of host configurations and benign workloads. Furthermore, the detector exhibited a high detection rate after having observed as few as $1{,}000$ system calls, corresponding to an average execution time of 205 ms. The system detected the execution of the malware samples during their early stages of execution, typically when they were covertly installing themselves and making other host modifications.

Acknowledgments

Thank you to the KEYSPOT Network, the People's Emergency Center, the Dornsife Center for Neighborhood Partnerships, the City of Philadelphia's Mayor's Commission on Literacy, Office of Innovation and Technology (OIT), and Department of Parks and Recreation (PPR) for their support of this research. This research is sponsored by a Secure and Trustworthy Cyberspace (SaTC) award from the National Science Foundation (NSF), under grant CNS-1228847.

References

[1] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Computing Surveys*, 2008.

[2] S. Forrest, S. Hofmeyr, A. Somayaji, and T. Longstaff, "A sense of self for unix processes," in *IEEE Security and Privacy*, 1996.

[3] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *Annual Computer Security Applications Conference*, ACSAC, 2007.

[4] N. Ye, X. Li, Q. Chen, S. M. Emran, and M. Xu, "Probabilistic techniques for intrusion detection based on computer audit data," *IEEE Trans. Syst., Man, Cybern. A*, 2001.

[5] R. Moskovitch, Y. Elovici, and L. Rokach, "Detection of unknown computer worms based on behavioral classification of the host," *Computational Statistics and Data Analysis*, 2008.

[6] A. Moser, C. Kruegel, and E. Kirda, "Exploring multiple execution paths for malware analysis," in *IEEE Symposium on Security and Privacy*, 2007.

[7] A. Slowinska and H. Bos, "Pointless tainting?: evaluating the practicality of pointer tainting," in *European conference on computer systems*, EuroSys, 2009.

[8] M. Christodorescu, S. Jha, S. A. Seshia, D. Song, and R. E. Bryant, "Semantics-aware malware detection," *IEEE Symposium on Security and Privacy*, 2005.

[9] D. Bruschi, L. Martignoni, and M. Monga, "Detecting self-mutating malware using control-flow graph matching," in *Detection of Intrusions and Malware & Vulnerability Assessment*, Lecture Notes in Computer Science, 2006.

[10] J. Kinder, S. Katzenbeisser, C. Schallhart, and H. Veith, "Proactive detection of computer worms using model checking," *IEEE Trans. Dependable Secure Comput.*, 2010.

[11] Q. Zhang and D. Reeves, "Metaaware: Identifying metamorphic malware," in *Annual Computer Security Applications Conference*, 2007.

[12] D.-K. Kang, D. Fuller, and V. Honavar, "Learning classifiers for misuse and anomaly detection using a bag of system calls representation," in *Annual Information Assurance Workshop*, 2005.

[13] Y. Liao and V. R. Vemuri, "Using text categorization techniques for intrusion detection." in *USENIX Security Symposium*, 2002.

[14] B. Mehdi, F. Ahmed, S. Khayyam, and M. Farooq, "Towards a theory of generalizing system call representation for in-execution malware detection," in *International Conference on Communications*, ICC, 2010.

[15] J. Pfoh, C. Schneider, and C. Eckert, "Leveraging string kernels for malware detection," in *Network and System Security*, Lecture Notes in Computer Science, 2013.

[16] C. Warrender, S. Forrest, and B. Pearlmutter, "Detecting intrusions using system calls: alternative data models," in *IEEE Symposium on Security and Privacy*, 1999.

[17] H. Xiao and T. Stibor, "A supervised topic transition model for detecting malicious system call sequences," in *workshop on Knowledge discovery, modeling and simulation*, 2011.

[18] S. Axelsson, "The base-rate fallacy and the difficulty of intrusion detection," *ACM Transactions on Information System Security*, 2000.

[19] A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu, and E. Kirda, "Accessminer: using system-centric models for malware protection," in *Conference on Computer and Communications Security*, CCS, 2010.

[20] 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 *International Symposium on Software Testing and Analysis*, ISSTA, 2012.

[21] M. A. H. Ian H. Witten, Eibe Frank, *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2011.

[22] A. M. Kibriya, E. Frank, B. Pfahringer, and G. Holmes, "Multinomial naive bayes for text categorization revisited," in *AI 2004: Advances in Artificial Intelligence*, Lecture Notes in Computer Science, 2005.

[23] H. L. VanTrees, *Detection, Estimation, and Modulation Theory*. Wiley-Interscience, 2001.

[24] T. Zhang, "Solving large scale linear prediction problems using stochastic gradient descent algorithms," in *International Conference on Machine Learning*, ICML, 2004.

[25] A. Genkin, D. D. Lewis, and D. Madigan, "Large-scale bayesian logistic regression for text categorization," *Technometrics*, 2007.

[26] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Machine learning*, 2002.

[27] E. Leopold and J. Kindermann, "Text categorization with support vector machines. how to represent texts in input space?" *Machine Learning*, 2002.