Malware Detection using Behavioral Whitelisting of Computer Systems

Saumya Saxena
Department of Computer Science
Drexel University
Philadelphia, PA USA
ss4554@drexel.edu

Spiros Mancoridis
Department of Computer Science
Drexel University
Philadelphia, PA USA
mancors@drexel.edu

***Abstract*—Malware detection has been an active area of research for a long time. With the rapid growth of self-mutating malware, many malware-detection tools fail quickly or have a high rate of false positives. Our work tackles the problem differently by creating anomaly detectors for computer systems. Since the number of potential malware far exceeds the number of benign software on any given computer system, our thesis is that it is possible to efficiently detect malware as anomalies in the expected behavior of computer systems hosting only benign software. This is in contrast to traditional approaches that attempt to construct behavioral models for every possible instance or type of malware.**

***Index Terms*—Behavioral Malware Detection, Whitelisting, Machine Learning, Computer Systems**

I. Introduction

Malware detection refers to the process of determining the presence of malware on computer systems. The two broad approaches used to detect malware can be labeled as static or dynamic. The static approaches create a signature for each malware sample using features such as the binary sequences in executable files. The dynamic approaches look for behavioral "footprints" the malware leave behind when they execute on infected computer systems.

Static malware detection requires examining malware to create a distinct signature for each newly discovered malware. A signature can be based on a byte-code sequence [15], assembly language instruction sequence [16], list of imported dynamic link libraries [20], or function call names found in the code [3], [19]. Unfortunately, malware authors use various obfuscation techniques to generate new variants of the same malware [22]. Therefore, the number of static signatures needed to detect all known malware grows rapidly, as does the time it takes to analyze each malware sample to create its signature. This endangers critical systems and increases the spread of malware infection.

The drawbacks of static malware detectors led to the burgeoning research area of behavioral malware detection, where the behavioral execution patterns of malware are captured as machine learning models. Despite the large number of malware variants, the original malware tend to exhibit similar execution behavior [18], [23], as static obfuscation techniques often do not change the behavior of the malware. Therefore, examining the dynamic properties of the malware is more reliable than examining easily obfuscated static properties.

The caveat of behavioral approaches is that extracting informative dynamic properties from soft wares' behavior is complicated. The computing environment needs to create the right conditions to execute malware. For example, the environment needs to have the same set of configurations (i.e., the specific vulnerability type in the application, the same version of the operating system) to activate the malicious behavior of the malware. Malware also have different behaviors depending on the conditions of the infected machine.

Our research attempts to construct behavioral models of computer systems that host benign software exclusively, rather than attempt to create models to detect all of the possible malware that can infect the system. This is, in effect, a behavioral

This research was funded by the Auerbach Berger Chair in Cybersecurity held by Spiros Mancoridis.

978-1-7281-5092-5/19/$31.00 ©2019 IEEE

whitelisting approach to malware detection, which is different from the traditional way of viewing antivirus, in general.

We use sequences of kernel-level system calls made by the computer system at run-time as the only behavioral feature to construct our models. We then feed the data from this feature to a 1-class SVM [9] that distinguishes whether system call sequences are "normal" sequences made by a computer system running only benign software, or "abnormal" sequences generated from the execution of malware.

We created a labeled data set by recording the sequence of system calls made by individual software instances in a clean computer system that was free of malware infections. We then repeated this process after infecting the same system with malware. We used the system call sequences gathered from the benign system to train a detector, based on a SVM, that can determine if a computer system is executing benign software exclusively, or not.

The rest of the paper is structured as follows: Section II describes related prior research; Section III describes the testbed created to run our experiments; Section IV details how labeled datasets were created for two distinct computer systems, one running a MySQL server and another running an Apache server, where the systems were executed with and without the presence of malware; Section V describes the machine learning technique that was used to create anomaly detectors for each computer system using the labeled datasets and machine learning; Section VI lists the results from our experiment and scores the effectiveness of our technique to differentiate computer systems that are free of malware from those that are infected my malware; and, finally, Section VII summarizes the conclusions of our work and identifies opportunities for further research.

II. Related Work

Behavioral malware detection has been proposed to overcome the limitations of the static signature-based malware detection [13]. Such detection captures the run-time behavior of malware during its execution. Behavioral malware detection relies on various dynamic properties as features such as a file system activities [21], terminal commands [24], network communication [25], and system calls [14], [26]. The malware detectors in previous work were designed as either a set of rules, or learned using machine learning algorithms. Our own previous work explored dynamic features such as system calls [5] and Microsoft Windows Prefetch Files [4].

The work described herein aims to accomplish the behavioral detection of malware using the whitelisting of computer systems that exclusively host benign software. The detector is created using system call patterns fed into a 1-class SVM. The technique is tested by deploying malware on computer systems that were originally devoid of malware and observing if the SVM model identifies anomalous system call sequences.

The prior art emphasizes building detectors to determine behavioral signatures of known malware, or of malware that is similar in behavior to pre-existing malware. Our work is different in that it emphasizes building detectors of computer systems composed exclusively of benign software (the inverse problem, if you like) and using anomaly detection to infer the presence of malware. Special cases of this approach are described in our recent previous work as it pertains to the specialized computer domains of routers [6] and Alexa-enabled Intelligent Assistants [7].

III. Experimental Testbed

We created a testbed to support our experiments. The idea was to create a testbed from which we could extract sequences of operating system kernel calls generated from the execution of two distinct computer systems, one running a MySQL server [17] and another running an Apache server [8]. After these data were gathered for each of the two systems, we repeated the same process, but first infecting each computer system with malware that were co-executing with benign MySQL and Apache servers, respectively.

The following software and tools were installed on our testbed.

- MySQL server
- Apache server
- Virtual Box
- MySQLslap
- Apache JMeter
- Fakenet-NG

978-1-7281-5092-5/19/$31.00 ©2019 IEEE

- VirusShare
- Heimdall Sensors

The testbed was generated on Virtual Box with two virtual machines, one to obtain clean data (i.e., traces of benign computer systems) and another to obtain infected data (i.e., traces of infected computer systems). The virtual machines ran the Ubuntu 16.04 LTS operating system. Each virtual machine had 30GB memory and 16GB RAM. We installed MySQL server version 14.14 Distribution 5.7.25 and Apache version 2.4.18 on each virtual machine.

The MySQLslap and Apache JMeter tools were used as workload emulators for MySQL and Apache servers, respectively. Fakenet-NG [10] was used to emulate an artificial network connection so malware samples could execute on the testbed without infecting other hosts on our network. This is essential because many of the malware samples we obtained from VirusShare quit immediately if they determine that they are not connected to a network. Finally, we used the Heimdall Sensors to extract system call traces made to the operating system kernel by each computer system.

IV. Data Generation

We had to exercise a multitude of capabilities of MySQL and Apache in order to generate realistic workloads so we could, in turn, extract system call traces that could be used to build benign, non-anomalous, behavioral models. An extensive list of the software capabilities of each benign program in our testbed, namely MySQL and Apache, were exercised to generate system call sequences that could be used to characterize their normal execution. We used the Heimdall [12] tool to record the system calls made during the execution of our experiments.

A. Data Generation from MySQL

Generating the behavioral system call trace data for MySQL involved two python scripts. One script was executed only once to create the tables in the database and insert values into the tables. The other script was executed to perform a MySQLslap query to exercise the capabilities of the MySQL server, such as, for example, executing `select` and `join` queries. All of these tasks were performed while running the Heimdall sensors to record the system calls being made to the operating system.

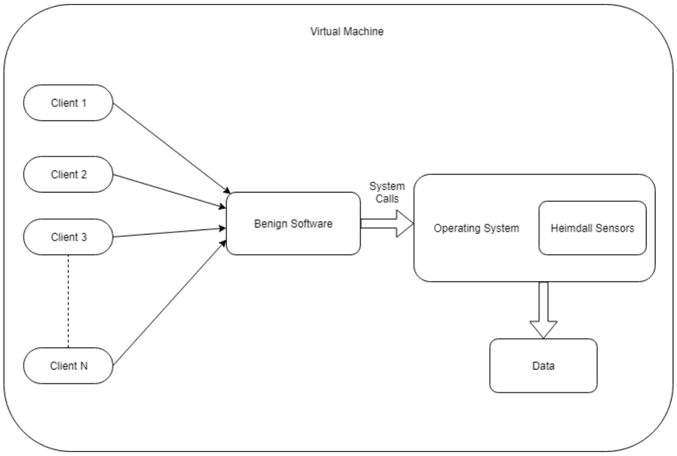

Fig. 1. Data Generation

We created six SQL tables. Two primary tables, one that had numerical data and another that had string data. We then used these two tables to derive four more tables, such that, all the odd valued IDs went into one table and the even ones went into the other table. Doing this for both numeric and string data, produced a total of six tables on which we could perform our SQL operations. This was done because, in order to access table joins, we needed to have a foreign key that could be used to map one table's data to another. This way we were be able to exercise more capabilities of MySQL. To fill in the values into the tables, we used a function that inserts random numeric and string values into the table and then creates the other four tables form these tables.

B. Data Generation from Apache

After obtaining data for MySQL, we started a similar process for Apache. We first created an html web page to be hosted on the Apache server, namely, `index.html`. Then, we placed this file in the `/var/www/html` directory. Finally, we used a web browser to connect to `127.0.0.1/index.html` in order to access the page on the server.

To emulate the client load in this case, we used a third-party application known as Apache JMeter. To do that, we created a Thread Group element that tells JMeter the number of users we wanted to simulate, how often the users would send requests, and how many requests they would send. We then edited the HTTP request properties, which defined

978-1-7281-5092-5/19/$31.00 ©2019 IEEE

the tasks that the users would perform. We executed JMeter along with the Heimdall sensors to record the system calls being made to the operating system during our experiment.

C. Data Generation from MySQL with Malware

To infect our system with malware, we downloaded malware from www.virusshare.com. Our testing set had 9484 malware samples. Since our testing samples were very generic malware written for the Linux operating system, many of the samples did not execute properly and, hence, did not produce any system call sequences. For each experiment, we selected a few malware samples from the samples that we were able to execute. We started generating the data by executing the malware simultaneously with MySQL. To do that, we first started Fakenet-NG to protect our local machines. Then, we started the Heimdall sensors. Finally, we ran the malware along with the MySQL server and its respective workload emulator. We recorded the system calls made by the benign and malicious software.

D. Data Generation from Apache with Malware

Once we generated the system call data for MySQL with malware, we similarly started generating the data by executing the malware simultaneously with Apache. To do that, we first started Fakenet-NG. Then, we started the Heimdall sensors. Finally, we ran the malware simultaneously with the Apache server and its respective workload emulator. We recorded the system calls made by the benign and malicious software.

V. Data Analysis

The output of the data generation task produced two files, `MySQL.log` and `apache.log` for both clean and infected systems. The log files consisted of comma-separated 4-tuples, namely, the name of the process, the process ID, the time-stamp at which the system call to the OS kernel was generated and the system call number. To input these to the 1-class SVM algorithm, we only need the system call number, hence, we extracted the last column into a separate list from the structure of these files.

Since the log files become very large quickly (as they are recording all the system calls made to the operating system), we had to generate 20 log files for each clean and infected computer systems.

A. Generating N-Grams

Each system call trace is a sequence of system calls, and every system call can be uniquely identified by a system call number, which is a positive integer. As an essential representation in natural language processing, the bag-of-n-grams model can be used to represent a system call trace as a sum of the one-hot vectors of n-grams appearing in the trace. A n-gram is a sequence of system call numbers appearing in a small window of length n in a trace [6].

For the purpose of our experiment, we took individual system calls (1-gram), sequences of two consecutive system calls (2-gram), and sequences of 3 consecutive system calls (3-gram). The following steps are repeated for each of the 1-gram, 2-gram, and 3-gram cases as follows:

- Declare dictionaries for system calls for benign data and infected data.
- Populate the dictionaries with individual system calls. The keys refer to the individual system calls and the values are the frequency of those calls in the log files.
- Create an array of system calls and apply the 1-class SVM model using benign data for training and a mixture of benign and infected data for testing.
- Predict the target values and evaluate the accuracy.

B. 1-class SVM

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyper-plane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyper-plane that categorizes new examples. In two-dimensional space this hyper-plane is a line dividing a plane in two parts [7].

1-class SVM is a special SVM that acts as an anomaly detector. It separates benign data from anomalous data. The 1-class SVM finds a hyper-plane that separates the training data from the origin with a margin that is as large as possible. Its objective function minimizes the normalized weights vector w of the hyper-plane (which is equivalent to maximizing the margin), and the objective function is also penalized by points that lie on the wrong

978-1-7281-5092-5/19/$31.00 ©2019 IEEE

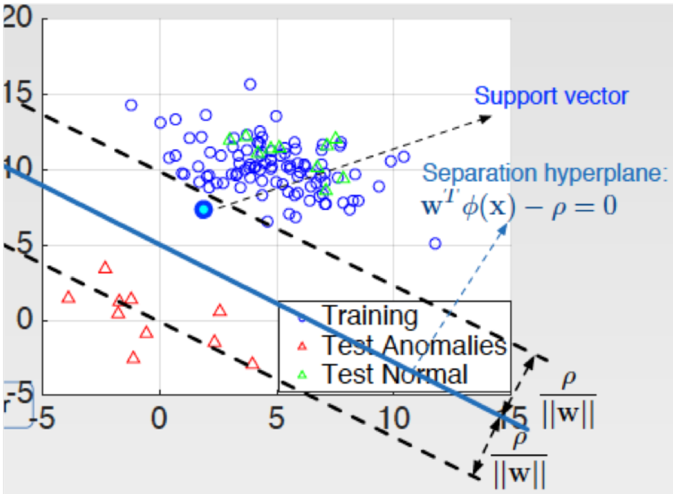

Fig. 2. Illustration of a 1-class SVM that separates benign features from malicious ones

side of the hyper-plane (i.e., the side wherein the origin lies). The test statistic of the 1-class SVM is the distance to the separating hyper-plane.

While creating our anomaly detector, we initially create n-grams of benign execution traces for the purpose of training our 1-class SVM model and then, we mix benign and anomalous traces when testing our model for anomalies encountered during testing that were not observed during training.

VI. Evaluation

The results of the experiment show that as we increase the length of the system call sequences, the false positive rate approaches to 0. The true positive rate is equal to 1 regardless of the length of the system call sequences used in the training of the detector. This is not surprising because each computer system in our experiments performs a very specialized task, namely database management and web serving, respectively.

This work is compatible with trends in deploying software servers on virtual machines in the cloud. In such deployments, the virtual machine typically runs one, or a small number of, applications. It is, therefore, feasible to create behavioral models for virtual computer systems running a small number of applications rather than try to build models for all possible malware samples that may land on such virtual machines.

	1-gram	2-gram	3-gram
TPR	1.0	1.0	1.0
FPR	0.117	0.0	0.0

TABLE I
Accuracy of 1-class SVM for MySQL

	1-gram	2-gram	3-gram
TPR	1.0	1.0	1.0
FPR	0.125	0.0	0.0

TABLE II
Accuracy of 1-class SVM for Apache

where,

$$TPR(True\ Positive\ Rate) = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{1}$$

True Positive (TP) is the fraction of detecting a malicious sample as malicious. False Negative (FN) is the fraction of falsely detecting a malicious as a benign.

$$FPR(False\ Positive\ Rate) = \frac{False\ Positive}{False\ Positive + True\ Negative} \tag{2}$$

False Positive (FP) is the fraction of falsely detecting a benign sample as malicious. True Negative (TN) is the fraction of detecting a benign sample as benign.

True Positive Rate (TPR) measures the accuracy performance of malware detectors while, False Positive Rate (FPR) measures the performance fault of malware detectors. A successful malware detector has a high TPR.

Our approach relies on building behavioral models for computer systems that exclusively execute benign software and using them as anomaly detectors to determine the presence of malware on a system. The results exhibit that if we have a large enough dataset to train a 1-class SVM, whitelisting is a promising approach to detecting the presence of malicious code on computing systems.

VII. Conclusions

With the results observed, we demonstrated that building a behavioral model for each computer system (i.e., whitelisting) is a promising approach, especially in lieu of the threat of polymorphic and

978-1-7281-5092-5/19/$31.00 ©2019 IEEE

metamorphic malware and their adverse effect on existing malware detectors.

However, as is typical of new malware detection research, adversaries can adapt to the state-of-the-art and eventually defeat it. In this case, future malware authors would be encouraged to write malware that emulates the computer systems they are targeting while surreptitiously executing their malicious activity. A possible counter-counter measure from future malware detection researchers would be to train a generative adversarial neural network [11] that would be robust to stealthy malware.

Another line of research could be directed towards creating such feature sets patterns for operating systems other than Linux and determining the category and family (e.g., botnet, trojan) of a detected malware sample automatically.

References

[1] *12th International Conference on Malicious and Unwanted Software, MALWARE 2017, Fajardo, PR, USA, October 11-14, 2017*. IEEE Computer Society, 2017.

[2] *13th International Conference on Malicious and Unwanted Software, MALWARE 2018, Nantucket, MA, USA, October 22-24, 2018*. IEEE, 2018.

[3] Mamoun Alazab, Sitalakshmi Venkatraman, Paul Watters, and Moutaz Alazab. Zero-day malware detection based on supervised learning algorithms of api call signatures. In *Proceedings of the Ninth Australasian Data Mining Conference-Volume 121*, pages 171–182. Australian Computer Society, Inc., 2011.

[4] Bander Alsulami and Spiros Mancoridis. Behavioral malware classification using convolutional recurrent neural networks. In *13th International Conference on Malicious and Unwanted Software, MALWARE 2018, Nantucket, MA, USA, October 22-24, 2018* [2], pages 103–111.

[5] Bander Alsulami, Avinash Srinivasan, Hunter Dong, and Spiros Mancoridis. Lightweight behavioral malware detection for windows platforms. In *12th International Conference on Malicious and Unwanted Software, MALWARE 2017, Fajardo, PR, USA, October 11-14, 2017* [1], pages 75–81.

[6] Ni An, Alexander M. Duff, Gaurav Naik, Michalis Faloutsos, Steven Weber, and Spiros Mancoridis. Behavioral anomaly detection of malware on home routers. In *12th International Conference on Malicious and Unwanted Software, MALWARE 2017, Fajardo, PR, USA, October 11-14, 2017* [1], pages 47–54.

[7] Ni An, Alexander M. Duff, Mahshid Noorani, Steven Weber, and Spiros Mancoridis. Malware anomaly detection on virtual assistants. In *13th International Conference on Malicious and Unwanted Software, MALWARE 2018, Nantucket, MA, USA, October 22-24, 2018* [2], pages 124–131.

[8] Apache Software Foundation. https://httpd.apache.org.

[9] Bernhard E. Boser, Isabelle Guyon, and Vladimir Vapnik. A training algorithm for optimal margin classifiers. In David Haussler, editor, *Proceedings of the Fifth Annual ACM Conference on Computational Learning Theory, COLT 1992, Pittsburgh, PA, USA, July 27-29, 1992.*, pages 144–152. ACM, 1992.

[10] Fakenet-NG. https://github.com/fireeye/flare-fakenet-ng.

[11] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2672–2680, 2014.

[12] Heimdall Operating System Kernel Call Tracer. https://github.com/ronin-zero/heimdall.

[13] Grégoire Jacob, Hervé Debar, and Eric Filiol. Behavioral detection of malware: from a survey towards an established taxonomy. *Journal in computer Virology*, 4(3):251–266, 2008.

[14] Dae-Ki Kang, Doug Fuller, and Vasant Honavar. Learning classifiers for misuse and anomaly detection using a bag of system calls representation. In *Information Assurance Workshop, 2005. IAW'05. Proceedings from the Sixth Annual IEEE SMC*, pages 118–125. IEEE, 2005.

[15] J Zico Kolter and Marcus A Maloof. Learning to detect and classify malicious executables in the wild. *Journal of Machine Learning Research*, 7(Dec):2721–2744, 2006.

[16] Robert Moskovitch, Clint Feher, Nir Tzachar, Eugene Berger, Marina Gitelman, Shlomi Dolev, and Yuval Elovici. Unknown malcode detection using opcode representation. In *Intelligence and Security Informatics*, pages 204–215. Springer, 2008.

[17] MySQL. https://www.mysql.com/.

[18] Carey Nachenberg. Computer virus-coevolution. *Communications of the ACM*, 50(1):46–51, 1997.

[19] A-D Schmidt, Rainer Bye, H-G Schmidt, Jan Clausen, Osman Kiraz, Kamer A Yuksel, Seyit Ahmet Camtepe, and Sahin Albayrak. Static analysis of executables for collaborative malware detection on android. In *Communications, 2009. ICC'09. IEEE International Conference on*, pages 1–5. IEEE, 2009.

[20] Matthew G Schultz, Eleazar Eskin, F Zadok, and Salvatore J Stolfo. Data mining methods for detection of new malicious executables. In *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*, pages 38–49. IEEE, 2001.

[21] Salvatore J Stolfo, Frank Apap, Eleazar Eskin, Katherine Heller, Shlomo Hershkop, Andrew Honig, and Krysta Svore. A comparative evaluation of two algorithms for windows registry anomaly detection. *Journal of Computer Security*, 13(4):659–693, 2005.

[22] Annie H Toderici and Mark Stamp. Chi-squared distance and metamorphic virus detection. *Journal of Computer Virology and Hacking Techniques*, 9(1):1–14, 2013.

[23] Michael Venable, Andrew Walenstein, Matthew Hayes, Christopher Thompson, and Arun Lakhotia. Vilo: a shield in the malware variation battle. *Virus Bulletin*, pages 5–10, 2007.

[24] Ke Wang and Salvatore Stolfo. One-class training for masquerade detection. 2003.

[25] Nong Ye and Qiang Chen. An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems. *Quality and Reliability Engineering International*, 17(2):105–112, 2001.

[26] Nong Ye, Xiangyang Li, Qiang Chen, Syed Masum Emran, and Mingming Xu. Probabilistic techniques for intrusion detection based on computer audit data. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 31(4):266–274, 2001.

978-1-7281-5092-5/19/$31.00 ©2019 IEEE