Identification of Program Signatures From Cloud Computing System Telemetry Data

Nicole Nichols, Mark Greaves, William Smith
Pacific Northwest National Laboratory
1100 Dexter Ave N, Suite 400
Seattle, WA 98109

Ryan LaMothe, Gianluca Longoni, Jeremy Teuton
Pacific Northwest National Laboratory
902 Battelle Boulevard
Richland, WA 99352

***Abstract*—Malicious cloud computing activity can take many forms, including running unauthorized programs in a virtual environment. Detection of these malicious activities while preserving the privacy of the user is an important research challenge. Prior work has shown the potential viability of using cloud service billing metrics as a mechanism for proxy identification of malicious programs. Previously this novel detection method has been evaluated in a synthetic and isolated computational environment.**

In this paper we demonstrate the ability of billing metrics to identify programs, in an active cloud computing environment, including multiple virtual machines running on the same hypervisor. The open source cloud computing platform OpenStack, is used for private cloud management at Pacific Northwest National Laboratory. OpenStack provides a billing tool (Ceilometer) to collect system telemetry measurements. We identify four different programs running on four virtual machines under the same cloud user account. Programs were identified with up to 95% accuracy. This accuracy is dependent on the distinctiveness of telemetry measurements for the specific programs we tested.

Future work will examine the scalability of this approach for a larger selection of programs to better understand the uniqueness needed to identify a program. Additionally, future work should address the separation of signatures when multiple programs are running on the same virtual machine.

I. Introduction

According to an article by Neal Leavitt for the IEEE computer society [1], cloud computing is becoming one of the fastest growing specialties in information technology. However, many industry experts have voiced concerns over the unique security and privacy challenges that occur in large clouds. A key report by the Cloud Security Alliance describes the top ten security challenges for this type of infrastructure [2]. Other key publications provide additional background [3], [4], [5], [6], [7].

The resonant themes of these papers include:

1) Data security - Breaches of data security can occur through a variety of mechanisms including theft or loss due to drive failures and inadequate backup.
2) Nefarious use of cloud services - Cyber criminal activities take advantage of the anonymity of cloud services, using them to host spammers, malicious code, botnets, and phishing websites.
3) Malicious insiders - Employees of cloud providers have potential access to private customer data. Without adequate monitoring and breach notification policies, this activity could go undetected.
4) Insufficient isolation of virtual machines - Several attack methods exploit vulnerabilities between tenant virtual machines

In the presented research, we look specifically at the problem of cloud providers detecting malicious programs running on their cloud. Privacy laws often prevent detailed inspection of cloud users activity, however several cloud company executives have noted this as a significant problem for their businesses [8].

II. Prior Work

In the field of cloud security, there are a few specific research topics and considerations that inform our approach to detecting malicious programs running in a cloud environment:

1) Malware detection
2) Intrusion detection
3) System noise and background activity

A. Malware

There is significant prior work in malware detection, both for cloud computing and for traditional computing infrastructure [9], [10], [11]. Regardless of the domain, cloud or traditional computing, the methods for performing malware detection can be grouped by similar frameworks. Yuxin et al. provide a comprehensive interpretation of such a framework [10]. At the top level, methods are divided into two categories, signature-based and behavior-based detection methods.

Signature based approaches typically analyze binary instructions and use regular expressions or statistical models of codeword frequency to predict if a program matches the signatures of known malicious programs. Signature based methods can be very accurate, but they are confusable, brittle, and do not generalize well. In the context of the cloud domain, these methods also require privileged access to memory in order to make a prediction, which would violate our goal of privacy preservation for the end-user. Privacy, in the context of cloud computing, is typically discussed in relation to data auditing, to ensure the correctness of the data and privacy preserving techniques do so without direct access to the data [12].

Behavior based approaches model a program's behavior, and behaviors are actions initiated by the program such as opening a file, writing a log file, changing user permissions, etc. Behavior based methods can be refined into three sub-categories: static behavior detection, dynamic behavior detection, and heuristic behavior detection. The heuristic based approach, [9], uses expert-defined rules to classify specific behaviors as malicious or benign. This can provide some effectiveness for detecting previously unseen malware, but only for previously used or specifically foreseen attack behaviors.

Dynamic behavior based methods [13], [14] analyze system calls for specific execution patterns of the program code. This approach is typically implemented in a controlled environment or a cloud virtual machine (VM), to protect the end-user. The primary drawback is the limited number of test cases it can execute.

Static behavior based analysis evaluates all possible execution paths of a program, without actually running the program. Because of the exhaustive nature, it will be guaranteed to trigger the malicious actions. However, it is not guaranteed that malicious program actions will be identified as such, it is computationally expensive, and feature representation is difficult. Shabtai et al. [9] and Yuxin et al. [10], provide additional detail on static methods.

Some aggregate systems combine both signature and behavior based methods [11]. Other more recent work utilizes machine learning techniques to identify anomalies [15], [16] or combine statistical learning with static and dynamic code analysis [17]. The research presented in this paper is also a hybrid, however we present signatures of program behaviors, rather than binary instructions of the source code. Additionally, our features are derivatives of actual program behavior, in order to preserve privacy of VM users.

B. Intrusion Detection

The task of intrusion detection is subtly different from malware detection, but malware can also be a precursor to intrusion. Both fall under the umbrella of cloud security and can have similar techniques, but the underlying actions to be detected are different. Because of the overlap in methodology used for intrusion detection it is worth reviewing intrusion detection, as some of the same techniques can be applied to malicious program detection.

The work of Modi et al. [18] and Patel et al. [19] provide comprehensive discussions of types of intrusion techniques as well as methods for both prevention and detection of these events. Intrusion prevention is an important aspect of security, but for the purposes of our discussion, intrusion detection is more relevant as they also use signature and behavior based techniques. Some of the intrusion methods are common between traditional and cloud computing environments. For example, root attacks and port scanning are common to both, while hypervisor based attacks are unique to the cloud.

One proposed approach to intrusion detection is virtual machine introspection [20]. This method leverages the virtual machine monitor, which has access to all the states and registers of the virtual machines running in a particular cloud environment. A set of signature detectors and policy modules are then used to detect intrusions based on the machine state.

C. System Noise

The noisy neighbor phenomenon is the impact of other tenants and virtual machines have on each other when residing on the same hypervisor. This is normally discussed in the context of system performance when trying to run large scale high performance (memory or cpu intensive) experiments in the cloud. However, this impact can additionally lead to cyber attacks as described by Varadarajan et al. [21].

III. Summary of Paper Contribution

This work demonstrates the ability of simple, privacy preserving metrics to identify potentially malicious programs. Earlier work [22], [7] have made related contributions, but our contribution is differentiated by its explicit focus on the impact of noise. Our data was collected from the Pacific Northwest National Laboratory private cloud computing environment. Two data collection configurations were compared: first, when programs are isolated on a single virtual machine and hypervisor; and second, when multiple virtual machines run concurrently on the same hypervisor. The experiments presented here are a first test of this methodology when realistic noise is included in the data collection process.

Two earlier papers present similar methodology for program and cyber attack recognition, however data from both were collected in more isolated environments. The first paper [7] emphasized detection of the five most common attacks as defined by the Cloud Security Alliance [2]: denial of service (DoS), cross VM side channel attack (CVMSC), malicious insider (MI), attacks targeting shared memory (ATSM), and phishing attacks (PA). Each of these scenarios are represented by specific actions on isolated server hardware. Details of their data collection can be found in [7].

Earlier work by Solanas et al. [22] demonstrated the ability of this technique to identify specific programs, in addition to attack types. However, their data collection system, though accurate and well described, was a noise-less system composed of three Intel NUC's networked together and run in isolation. A NUC (Next Unit of Computing) is a consumer grade micro-PC for streaming media. In contrast, our work reproduces the successes of this approach while systematically increasing the sources of system noise by operating on a real operational cloud environment and with multiple VM's on a single hypervisor.

IV. Experiment Setup

A. Hardware and System Configuration

The data obtained from these experiments was collected from an institutional private cloud computing system at Pacific Northwest National Laboratory running OpenStack (Kilo release). The underlying physical compute nodes that are exposed via a hypervisor are dual socket AMD Opteron 6272 CPUs, with each CPU consisting of 16 physical cores, 2.1

GHz clock, 512 KB of L3 cache memory, and 64GB RAM per socket (128GB total per node). Local physical storage consists of 2TB of RAID 1 disk.

The computing hardware exposed in each virtual machine consisted of eight AMD Opterons, with a 2.1 GHz clock and 512 KB of L3 cache memory, and a total of 16 GB of RAM per node. The cloud VMs are interconnected via 40GB IP over InfiniBand.

B. *Programs and Data Collection*

We chose an initial set of programs as proxies for known malicious programs (FFmpeg [23], Boltzmann [24], Cambridge Stars [25], and MSGF+ [26]). These programs were selected because they each represent a type of computation that may realistically be processed in batch, using cloud computing resources. These programs are all scientific simulation programs with the exception of FFmpeg which is an audio/video processing utility. Each program was scripted to run in loop for a designated time, using standard example files as input.

OpenStack is the cloud infrastructure management tool and the built-in Ceilometer system was used to collect real-time telemetry from the virtual machines. Ceilometer metrics are traditionally used to process billing. The sampling period of Ceilometer was set for all experiments to 5 seconds. Of the available meters, seven were used as input features for classification:

- CPU utilization
- disk.read.bytes.rate
- disk.read.requests.rate
- disk.write.bytes.rate
- disk.write.requests.rate
- network.incoming.bytes.rate
- network.outgoing.bytes.rate

All data was written to .csv log files that were processed off-line.

V. Results

For the following results, a base model was constructed for each program from 4 hours of data. When this base data was being collected, the program and its host VM were the only machine running on the hypervisor. A separate and independent collection of data was made for a noisy neighbor scenario. In this case, four VM's were running, one for each program and all on the same hypervisor. This noisy neighbor data was collected for 30 minutes and used to verify that the base models were still capable of recognizing programs in spite of potential system noise.

We used standard metrics to evaluate system performance: precision, recall, f-score and normalized confusion matrices. Two types of classification models, decision trees and K-nearest neighbor, were used to determine if model choice had a significant impact. Both models were implemented with the python package *scikit-learn* [27] specifically DecisionTreeClassifier and KNeighborsClassifier. The decision tree algorithm is an optimized version of CART [28], with a maximum tree depth of 45. The K-nearest neighbor method used K=2. These parameter values were chosen based on a grid search from related but unpublished work.

TABLE I
Scoring statistics for decision tree classifier

	Precision	Recall	F-Score
Boltzmann	.91	.95	.93
CS	.96	.92	.94
ffmpeg	1.0	1.0	1.0
MSGF+	.96	.96	.96

TABLE II
Normalized Confusion Matrix for decision tree classifier

	Boltzmann	CS	FFMPEG	MSGF+
True Boltzmann	0.95	0.02	0.00	0.03
True CS	0.07	0.92	0.00	0.01
True FFMPEG	0.00	0.00	1.0	0.00
True MSGF+	0.02	0.02	0.00	0.96

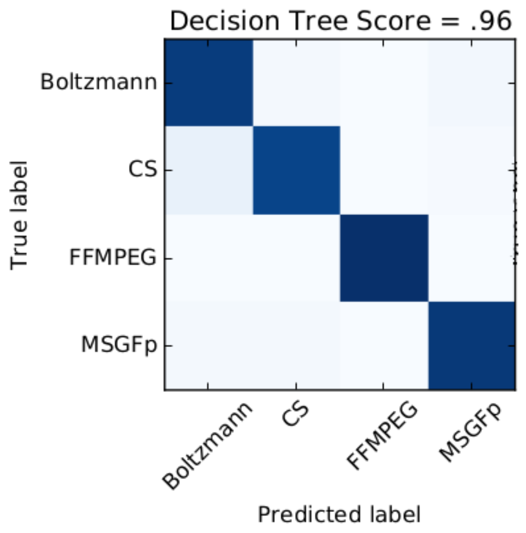

Fig. 1. Normalized Confusion Matrix for decision tree classifier

The performance metrics from the decision tree classification method are summarized in Table I. Precision and recall statistics across the programs range from 91% to 100%, with Boltzmann being the most difficult to predict and FFmpeg being the easiest to predict.

The normalized confusion matrix for the decision tree classifier is visualized in Figure 1 and the corresponding numeric values are summarized in Table II. From this table, we note that of the true Boltzmann samples, the most common classification error is MSGF+ (3%) and Cambridge Stars (2%). Of the true Cambridge Stars samples, the most common classification errors are Boltzmann (7%) and MSGF+(1%). For the MSGF+ program, it evenly splits the error between Boltzmann(2%) and Cambridge Stars (2%).

Performance metrics from the K-nearest neighbor classification method are summarized in Table I. Precision and recall

TABLE III
SCORING STATISTICS FOR K-NN CLASSIFIER

	Precision	Recall	F-Score
Boltzmann	.89	.96	.93
CS	.94	.91	.93
ffmpeg	1.0	1.0	1.0
MSGF+	1.0	.95	.98

TABLE IV
NORMALIZED CONFUSION MATRIX FOR K-NN CLASSIFIER

	Boltzmann	CS	FFMPEG	MSGF+
True Boltzmann	0.96	0.03	0.00	0.00
True CS	0.09	0.91	0.00	0.00
True FFMPEG	0.00	0.00	1.0	0.00
True MSGF+	0.03	0.02	0.00	0.95

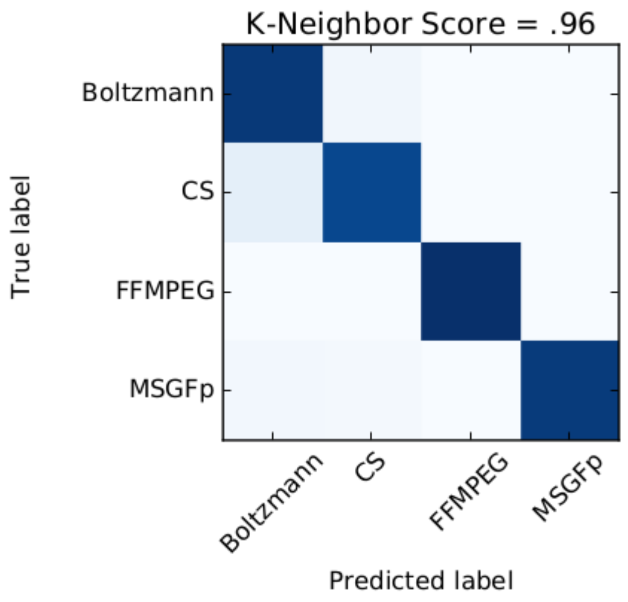

Fig. 2. Normalized Confusion Matrix for K-NN classifier

statistics across the programs ranged from 89% to 100%. Boltzmann is again the most difficult to predict and FFMPEG is the easiest to predict. One reason FFMPEG was selected as a program to identify was because it was easy to run continuously with variable input by compressing the audio of live radio streams. We suspect this same trait made the program easy to identify as it continuously used network I/O.

The normalized confusion matrix for the K-nearest neighbor classifier is visualized in Figure V and the corresponding numeric values are summarized in Table IV. The overall performance is nearly identical as that obtained for decision trees, however with the k-nearest neighbor approach, the incorrect prediction of Boltzmann and Cambridge Stars, as MSGF+ was eliminated. From Table IV, we note that of the true Boltzmann samples, the classification errors only occur as Cambridge Stars (3%). Of the true Cambridge Stars samples, the only classification errors occur as Boltzmann (9%). For the MSGF+ program, it splits the error between Boltzmann(3%) and Cambridge Stars (2%).

VI. DISCUSSION

Both K-NN and decision tree classification resulted in program detection statistics that were nearly identical, independent of which classification model was selected. Due to the high degree of separability of the signatures of these specific programs, we believe additional tuning of parameters, or more sophisticated models, would primarily result in overfitting. The promise of this work is the methodology, which can be applied to a larger and more diverse set of programs. Future work should investigate the scalability and limits of program signature uniqueness. Identification of multiple programs running on an individual VM could also be an interesting extension of this work.

VII. CONCLUSIONS AND FUTURE WORK

We demonstrate the ability of simple network statistics to reliably differentiate programs running individually on concurrent virtual machines on the same hypervisor. The collected metrics are privacy preserving, as they contain no identification of the user of the hypervisor.

There are many other mechanisms that occur in a real world environment which add noise to program signatures. For example, distributed programs running on a cluster, or multiple programs running on a single virtual machine. Future work will investigate the potential of this technique to differentiate programs in these more complex environments.

ACKNOWLEDGMENT

The authors would like to thank PNNL Institutional Computing (PIC) for infrastructure support. The research described in this paper was conducted under the Laboratory Directed Research and Development Program at PNNL, a multi-program national laboratory operated by Battelle for the U.S. Department of Energy.

REFERENCES

[1] Leavitt, Neal. ”Is cloud computing really ready for prime time.” Growth 27.5 (2009): 15-20.
[2] Alliance, Cloud Security. ”Top threats to cloud computing v1. 0.” (2010).
[3] Subashini, Subashini, and Veeraruna Kavitha. ”A survey on security issues in service delivery models of cloud computing.” Journal of network and computer applications 34.1 (2011): 1-11.
[4] Muttik, Igor, and Chris Barton. ”Cloud security technologies.” Information security technical report 14.1 (2009): 1-6.
[5] P. Balboni, K. Mccorry, and P.W David Snead, “Cloud Computing - benefits, risks, and recommendations for information security”, European Network and Information Security Agency, Tech. Rep. Nov 2009 http://www.enisa.europa.eu/act/rm/files/deliverables/cloud-computing-risk-assessment/.
[6] Wei, Jinpeng, et al. ”Managing security of virtual machine images in a cloud environment.” Proceedings of the 2009 ACM workshop on Cloud computing security. ACM, 2009.
[7] Khorshed, Md Tanzim, ABM Shawkat Ali, and Saleh A. Wasimi. ”A survey on gaps, threat remediation challenges and some thoughts for proactive attack detection in cloud computing.” Future Generation computer systems 28.6 (2012): 833-851.

[8] Grosse, Eric, et al. ”Cloud computing roundtable.” IEEE Security & Privacy 6.8 (2010): 17-23.
[9] Shabtai, Asaf, et al. ”Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey.” Information Security Technical Report 14.1 (2009): 16-29.
[10] Yuxin, Ding, et al. ”Feature representation and selection in malicious code detection methods based on static system calls.” Computers & Security 30.6 (2011): 514-524.
[11] Schmidt, Matthias, et al. ”Malware detection and kernel rootkit prevention in cloud computing environments.” 2011 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing. IEEE, 2011.
[12] Wang, Cong, et al. ”Privacy-preserving public auditing for data storage security in cloud computing.” INFOCOM, 2010 Proceedings IEEE. Ieee, 2010.
[13] Martignoni, Lorenzo, Roberto Paleari, and Danilo Bruschi. ”A framework for behavior-based malware analysis in the cloud.” International Conference on Information Systems Security. Springer Berlin Heidelberg, 2009.
[14] Oberheide, Jon, Evan Cooke, and Farnam Jahanian. ”CloudAV: N-Version Antivirus in the Network Cloud.” USENIX Security Symposium. 2008.
[15] Marnerides, Angelos K., et al. ”Malware detection in the cloud under Ensemble Empirical Mode Decomposition.” Computing, Networking and Communications (ICNC), 2015 International Conference on. IEEE, 2015.
[16] Watson, Michael R., et al. ”Malware detection in cloud computing infrastructures.” IEEE Transactions on Dependable and Secure Computing 13.2 (2016): 192-205.
[17] Zhang, Hanlin, et al. ”ScanMe mobile: a cloud-based Android malware analysis service.” ACM SIGAPP Applied Computing Review 16.1 (2016): 36-49.
[18] Modi, Chirag, et al. ”A survey of intrusion detection techniques in cloud.” Journal of Network and Computer Applications 36.1 (2013): 42-57.
[19] Patel, Ahmed, et al. ”An intrusion detection and prevention system in cloud computing: A systematic review.” Journal of network and computer applications 36.1 (2013): 25-41.
[20] T. Garfinkel, and M. Rosenblum, “A Virtual Machine Introspection Based Architecture for Intrusion Detection”
[21] Varadarajan, Venkatanathan, et al. ”Resource-freeing attacks: improve your cloud performance (at your neighbor’s expense).” Proceedings of the 2012 ACM conference on Computer and communications security. ACM, 2012.
[22] Solanas, Marc, Julio Hernandez-Castro, and Debojyoti Dutta. ”Detecting fraudulent activity in a cloud using privacy-friendly data aggregates.” arXiv preprint arXiv:1411.6721 (2014).
[23] http://ffmpeg.org/
[24] Cannon, William R. ”Simulating metabolism with statistical thermodynamics.” PloS one 9.8 (2014): e103582.
[25] http://www.ast.cam.ac.uk/ stars/
[26] http://omics.pnl.gov/software/ms-gf
[27] http://scikit-learn.org/stable/index.html
[28] Breiman, Leo, et al. Classification and regression trees. CRC press, 1984.