Project Summary

Malware is a group name given to many types of malicious software, it includes computer viruses, worms, rootkit, Adware, Trojan horses and spyware. Using antivirus softwares for malware detection is not robust since most antivirus applications act only on known and static byte patterns of malware. Detection of malicious software using hardware based detectors that implement special purpose registers is an emerging solution in security researches[1]. These registers are called Hardware Performance Counters (HPC). HPCs are integrated in modern microprocessors in order to count various micro architectural events such as cache misses and branch behavior.The main purpose of HPCS is to analyze and tune architectural level performance of running applications.

Most of the hardware based malware detectors use existing and new hardware performance counters to capture application execution patterns. Recent researches in security have focused on applying machine learning (ML) classifiers on data collected from these registers in order to classify the running applications as malware or benign with high level of accuracy. We implemented supervised training for each algorithm using a multitude of hardware performance counters to model execution patterns that scale a wide range of benign and malware applications. The effectiveness of these learning methods mainly relies on the information provided by the limited number of HPCs. Because of the limitation on the number of available hardware performance counters, ML classifiers can be used for making architectural decision on what counters HPCs are required to effectively improve ML classification accuracy[1].

In this project we first collected Performance Monitoring Unit (PMU) data for various malware and benign applications to build a comprehensive dataset. We then implemented various ML classification algorithms including Naive Bayes, SVM, Neural Networks and Random forest tree algorithms combined with ensemble techniques to select the best model for malware detection. We broke down the original comprehensive dataset into training and testing sets and analyzed the accuracy, true positive (TP) rate and complexity of various ML-classifier methods. We have used Linux PERF profiler tool[2] for collecting PMU data and Weka[3] as our ML workbench.

Introduction

Malware is a piece of software which is used by attacker to perform various malicious activities from stealing information and performing DoS attack to gaining root access. According to McAfee threats report [4] published in March 2016, 42 million malware samples have been recorded in the last quarter of 2015 alone with the rate of 316 new threats every minute.

Hardware Performance Counters (HPC) are special purpose registers, integrated in modern microprocessors in order to count various micro architectural events such as cache misses and branch behavior. The main purpose of HPCs is to analyze and tune architectural level performance of running applications[1]. ML classifiers are applied on data collected from HPCs in order to classify the running applications as malware or benign. HPC events are collected by running Linux perf tool while the application to be classified is getting executed in the CPU. Table 1 shows the typical micro architectural events that can be tracked in HPCs for Intel processors. ML classifiers demonstrated appealing accuracy and true positive (TP) results. By using the data collected form HPC a ML algorithm is used to detect potential malware codes, then a software mechanism is applied on all the instances that are detected as malware.Therefore, such a system should have high true

Table 1: List of events available in HPC

cpu-cycles	instructions	cache-references	cache-misses
branch-instructions	branch-misses	bus-cycles	ref-cycles
cpu-clock	task-clock	page-faults	context-switches
cpu-migrations	minor-faults	major-faults	alignment-faults
dummy	emulation-faults	L1-dcache-loads	L1-dcache-load-misses
L1-dcache-store-misses	L1-dcache-prefetch-misses	L1-icache-load-misses	LLC-loads
LLC-load-misses	LLC-stores	LLC-store-misses	LLC-prefetches
LLC-prefetch-misses	dTLB-loads	dTLB-load-misses	dTLB-stores
dTLB-store-misses	iTLB-loads	iTLB-load-misses	branch-loads
branch-load-misses	node-loads	node-load-misses	node-stores
node-store-misses	node-prefetches	node-prefetch-misses	L1-dcache-stores

positive rate for malware, in order not to miss any. On the other hand, low accuracy of the system can cause a huge overhead for the defense mechanism and reducing the efficiency of the overall system.

Background

To detect malware antivirus applications use byte signatures and patterns which are static characteristics of software. To avoid detection by antivirus application, malware developers use different obfuscation techniques such as encryption, polymorphism, metamorphism, dead code insertions, and instruction substitution in order to change the appearance of a file and its static characteristics. For example, it is possible to change hash sums used as file signatures (such that SHA-1 or md5) by simply changing different strings in the file. Further, dead code insertions can be used in executables to change opcode sequences, making detection troublesome[5].Several researches have shown anomalies associated with the execution behavior of malware software[6] [7], these anomalies include memory access patterns for instance. In this project we have used HPC data for modeling a ML classifier that can efficiently detect malware based on its execution behavior.

Problem Approach

We implemented and analyzed different machine learning algorithms to classify malware and benign applications in a Linux environment. We used Linux PERF to for collecting HPC values while applications are getting executed in CPU. PERF is a profiler tool for Linux 2.6+ based systems that abstracts away CPU hardware differences in Linux performance measurements and presents a simple command line interface. PERF is based on the perf_events interface exported by recent versions of the Linux kernel [8] and is the standard means of access to the hardware performance counters. PERF defines a set of common, symbolic hardware performance events. The number and kind of performance events is very specific to the machine micro-architecture[2] and are documented in processor manuals[9]. Typical HPC events available under Linux PERF tool for Intel Haswell processors are shown in Table 1.

We performed all data collections on a machine with an Intel Haswell Core i5-4590 CPU running Ubuntu 14.04 with Linux 4.4 Kernel. We used about 200 benign applications and more than 400 malware applications for HPC data collection using PERF tool. Benign applications consist of mibench benchmark suite[10], Linux system programs, browsers, text editors, and word processor. For malware applications, Linux malware are collected from virustotal.com[11]. Malware applications include Linux ELFs, python scripts, perl scripts, and bash scripts. Each HPC event in Table 1 represents a feature in the dataset. All collected events are transformed into vectors with events as columns, and extra column of "class" is added to each vector in which collected data is categorized as "malware" or "no malware" depending on the application type.

Binary classifiers are implemented for Malware detection with offline supervised training and testing procedures as shown in figure 1. We used WEKA ML workbench [3] for evaluation and analysis of the different ML classification algorithms and followed 70%-30% data set splitting for training and testing. We collected a total of 44 HPC events or features for each application execution profiled by PERF. Some of the events recorded in HPC are micro architectural behaviors that are descriptive of the platform where we collected the data, hence in order to make the classifier models platform architecture independent we removed those features from the dataset. Reducing the number of features also helps with significant reduction of offline learning time. A total of 16 features are used for all the classifier algorithms discussed in this project.

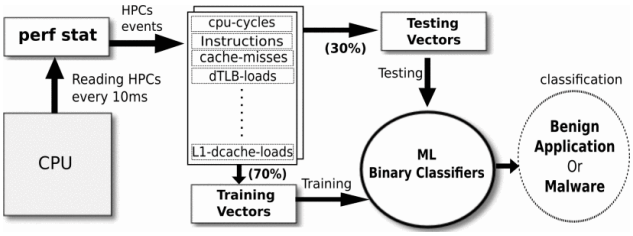

Figure 1: Offline training and testing procedure[1]

Table 2: ML algorithms implemented for malware detection

Classifier Algorithm	Parameters Used
SVM	Linear, Polynomial and rbf kernels & complexity (c)
Multiperceptron NN	hidden layer nodes
Bayes classifier	Naive Bayes
Logistic regression	-
Random forest	bagging and number of trees
Decision stumps	number of stumps

For each ML algorithm selected for this project as shown in table 2 we performed the following steps to analyze the performance and efficiency the classifiers :

1. Run the algorithm on training dataset using cross validation with 10 folds.
2. Record the complete summary of results for the algorithm and save the model.
3. Run the model developed once on the test dataset.
4. Implement voting for all the algorithms and analyze performance change

Results

Bayes Classifier

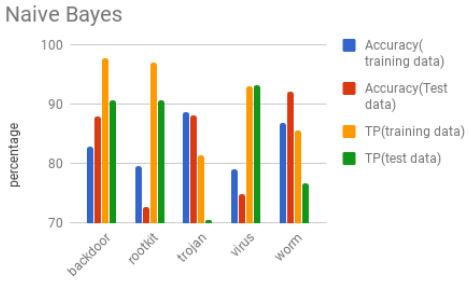

Figure 2: Accuracy and TP,Naive Bayes

Naive Bayes classifier models the posterior probabilities of classes based on the dataset we used for learning. Naive Bayes classifier assumes that all features are independent of each other but its noted the features used in this classification are somehow interdependent. This classifier has the best testing accuracy for Worms, while the true positive (TP) rate is 98% for Backdoors. TP for malware class rate is a measure of malware detection power, on the other hand, accuracy which considers TP and TN, can be a measure as detection power and overhead that is caused. Taking the two measurements into account the classifier performs best on Backdoors.

Logistic Regression

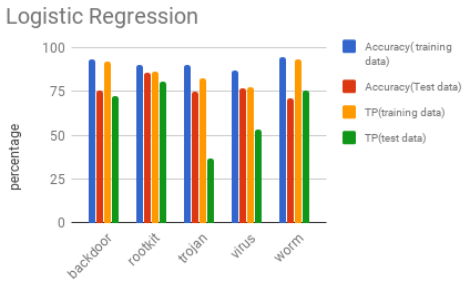

Figure 3: Accuracy and TP, Logistic Regression

Logistic Regression is a modification of Perceptron that has output of probability values between 0 and 1, rather than +1 or -1 as in the case of linear perceptron. Logistic regression employs gradient descent to find optimal weight values. Logistic Regression has the highest TP detection for Backdoor and Worms as compared to the other classes of malware.

Random Forest Bagging

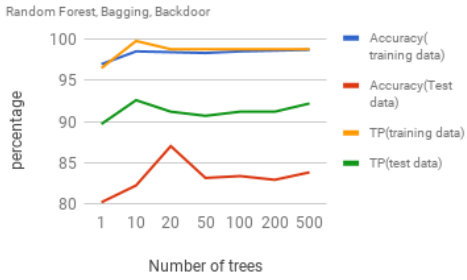

Figure 4: Accuracy and TP, Random Forest (Bagging)

Random forest is made of a large number of random trees. A random tree is formed by randomly choosing k features at each node and then finding the best threshold for that specific node, and going down till the tree is formed. K in our implementation is :

$$\lfloor(\log_2(\#offeatures) + 1)\rfloor$$

. The decision for the final class is made by voting decisions of each random tree. In our implementation depth of each random tree was in the order of 140.

Adaboost (Decision Stump)

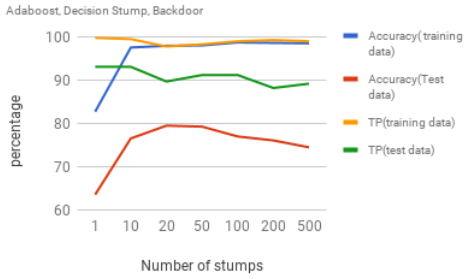

Figure 5: Accuracy and TP, Decision Stumps

We implemented Adaboost which incorporates decision Stumps. A decision Stump is a tree of depth 1. All of the stumps are trained based on the data, and their decision is weighted based on their accuracy. The final decision is the weighted average of all individual decisions. This algorithm is much faster to train in comparison to random Forest.

Support Vector Machines(SVM)

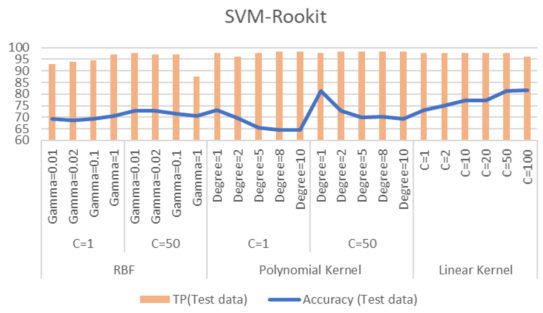

Figure 6: Accuracy and TP, SVM

We investigated different forms of SVM, using Linear, Polynomial, RBF kernels. Our results show that increasing complexity in SVM doesnt benefit us in terms of accuracy. Figure 6 shows increasing the degree of polynomial degraded accuracy. For polynomial kernel hard SVM(C=50) works better than soft SVM.For RBF kernel we can see the same trend, by increasing the Gamma, which means reducing complexity of the model we get better results in terms of accuracy. In the case of linear kernel increasing the complexity term, i.e. going towards hard SVM, resulted in better accuracy.

Neural Networks

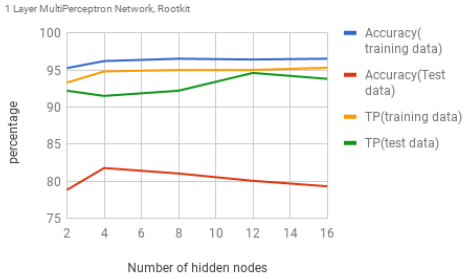

Figure 7: Accuracy and TP, NN

We implemented neural networks with only 1 hidden layer and the number of nodes in the hidden layer is increased 2 - 16, accuracy of the NN classifier is moderate with high overhead of false poistives.

Conclusion

In this project we have analyzed multiple classifiers performance for malware detection, by using HPC data as features describing the execution behavior of a software. The results observed during testing demonstrated appealing accuracy in malware detection. This investigation has shown each class of malware has it own unique execution pattern, where algorithms need fine tuning to different settings to obtain acceptable accuracy and TP rate. Overall Random Forest has shown the best performance (87%-95%) in terms of TP and Accuracy for all of the malware classes except worms. We have observed weaker performance for worms implementing ensemble voting (Figure 8) as compared to other classes of malware shown. Worms are generally small executables that replicate them selves and need more specific HPC configuration for better detection performance.

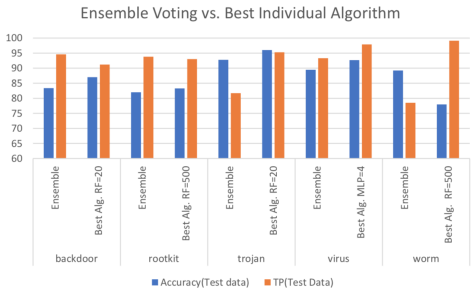

Figure 8: Accuracy and TP, NN

References

[1] N. Patel, A. Sasan, and H. Homayoun, "Analyzing hardware based malware detectors," in *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 2017, p. 25.

[2] P. J. Drongowski, "Perf tutorial: Counting hardware performance events." [Online]. Available: http://sandsoftwaresound.net/perf/perf-tut-count-hw-events/

[3] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[4] "Infographic: Mcafee labs threats report–march 2016," March 2016. [Online]. Available: https://www.mcafee.com/us/resources/misc/infographic-threats-report-mar-2016.pdf

[5] S. Banin, A. Shalaginov, and K. Franke, "Memory access patterns for malware detection," *Norsk informasjonssikkerhetskonferanse (NISK)*, pp. 96–107, 2016.

[6] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo, "On the feasibility of online malware detection with performance counters," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: ACM, 2013, pp. 559–570. [Online]. Available: http://doi.acm.org/10.1145/2485922.2485970

[7] A. Tang, S. Sethumadhavan, and S. J. Stolfo, "Unsupervised anomaly-based malware detection using hardware features," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2014, pp. 109–129.

[8] "Linux kernel profiling with perf." [Online]. Available: https://perf.wiki.kernel.org/index.php/Tutorial

[9] P. Guide, "Intel® 64 and ia-32 architectures software developer's manual," *Volume 3B: System programming Guide, Part*, vol. 2, 2011.

[10] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*. IEEE, 2001, pp. 3–14.

[11] "Virustotal intelligence service. http://www.virustotal.com/intelligence/. accessed: November 2016." [Online]. Available: http://www.virustotal.com/intelligence/

[12] X. Wang, C. Konstantinou, M. Maniatakos, and R. Karri, "Confirm: Detecting firmware modifications in embedded systems using hardware performance counters," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2015, pp. 544–551.

[13] M. Malik, S. Rafatirah, A. Sasan, and H. Homayoun, "System and architecture level characterization of big data applications on big and little core server architectures," in *2015 IEEE International Conference on Big Data (Big Data)*, Oct 2015, pp. 85–94.