Applying machine learning classifiers to dynamic Android malware detection at scale

Conference Paper · July 2013

DOI: 10.1109/IWCMC.2013.6583806

3 authors, including:

Brandon Amos
Carnegie Mellon University
24 PUBLICATIONS **558** CITATIONS
SEE PROFILE

Hamilton Allen Turner
Virginia Polytechnic Institute and State University
25 PUBLICATIONS **404** CITATIONS
SEE PROFILE

Applying machine learning classifiers to dynamic Android malware detection at scale

Brandon Amos, Hamilton Turner, Jules White
Dept. of Electrical and Computer Engineering, Virginia Tech
Blacksburg, Virginia, USA
Email:{bdamos, hamiltont, julesw}@vt.edu

***Abstract*—The widespread adoption and contextually sensitive nature of smartphone devices has increased concerns over smartphone malware. Machine learning classifiers are a current method for detecting malicious applications on smartphone systems. This paper presents the evaluation of a number of existing classifiers, using a dataset containing thousands of real (i.e. not synthetic) applications. We also present our STREAM framework, which was developed to enable rapid large-scale validation of mobile malware machine learning classifiers.**

***Keywords*—anomaly detection, machine learning, data collection, smartphones, mobile computing, IDS**

I. Introduction

Emerging Trends and Challenges. The International Data Corporation [1] states that Android's market share has grown to 136 million units and 75% of the market share in 3Q 2012. Bloomberg Businessweek [2] reports there are 700,000 applications available on Google Play as of October 29, 2012. While most of these applications are benign, TrendLabs reports in their 3Q 2012 security roundup that the top ten installed malicious Android applications had 71,520 installations total.

An important concern on the growing Android platform is malware detection. Malware detection techniques on the Android platform are similar to techniques used on any platform. Detection is fundamentally broken into *static* analysis, by analyzing a compiled file; *dynamic* analysis, by analyzing the runtime behavior, such as battery, memory, and network utilization of the device; or *hybrid* analysis, by combining static and dynamic techniques [3]. Static analysis is advantageous on memory-limited Android devices because the malware is not executed, only analyzed. However, dynamic analysis provides additional protection, particularly against polymorphic malware that change form during execution. To use the advantages from both static and dynamic analysis, desktop vendors such as AVG [4] employ hybrid techniques. Our work focuses specifically on *profiling* applications to obtain information used in dynamic analysis.

Behavior of both malicious and benign applications are profiled with a set of *feature vectors*, which are snapshots of system state information, such as memory utilization and power consumption. Machine learning algorithms are trained with known feature vectors to attempt to predict the classification of unknown feature vectors. Due to the range of hardware configurations a very large number of feature vectors from a diverse set of hardware are needed to effectively train machine learning algorithms. Future work will explore machine-invariant hardware metrics.

Gap in Research ⇒ Large-scale studies of the effectiveness of machine learning classifiers. A key open challenge is the lack of large-scale studies that use hundreds or thousands of mobile malware samples to analyze the effectiveness of well-known machine learning algorithms. To focus research and understand the effectiveness of current algorithms, studies are needed that provide empirical results from large-scale experiments. Key challenges include acquiring thousands of malware samples, orchestrating the install→profile→clean cycle, and training the machine learning classifiers.

Contribution ⇒ A large-scale study of malware classifiers and a distributed mobile malware experimentation platform.

To fill this gap in research on large-scale evaluations of machine learning algorithms for mobile malware detection, we present results from studying 6 machine learning classifiers on 1330 malicious and 408 benign applications, for a total of 1738 unique applications. By sending a large amount of emulated user input to each application, we collected 6,832 feature vectors. We analyzed classifier performance with cross validation (e.g. analysis performed on applications in the training set) and with true testing (e.g. analysis performed on applications not in the training set). Although cross validation frequently implies better malware classifier performance (e.g. is biased), it is used in many dynamic malware papers and therefore included here for comparison to prior results. The empirical results provide an initial roadmap for researchers selecting machine learning algorithms for mobile malware classification and a resource for validating the improvements of new malware classification approaches. Our results show that Logistic and Bayes net malware classifiers perform the worst and best respectively at detecting malware. Moreover, our datasets, empirical results, and distributed experimentation infrastructure are available in opensource form.

In addition to this study, we present the design of the distributed mobile malware experimentation framework that we created to profile applications and study malware classifiers: STREAM, a System for Automatically Training and Evaluating Android Malware Classifiers. STREAM is a feature vector collection framework that distributes collection to a cloud of Android devices or emulators. The feature vectors collected are generic and can be tuned to the needs of the user. The framework was designed to be configurable, allowing

978-1-4673-2480-9/13/$31.00 ©2013 IEEE

future researchers to reuse our automation framework instead of creating their own.

The remainder of this paper is organized as follows: Section II discusses the challenges faced during this study and while creating the framework; Section III covers STREAM, a rapid feature vector collection framework; Section IV presents empirical results from the study; and Section VI presents concluding remarks and lessons learned.

II. Challenges of Evaluating Mobile Malware Classifiers

While malware classifiers have the potential to detect and proactively prevent the spread of malware on mobile devices, there are a number of challenges to determining which techniques are most effective at detecting malware. A critical challenge is the need for the collection and experimentation with a large dataset for training malware classifiers, typically spanning hundreds of applications and thousands of feature vectors. These datasets can be difficult to collect accurately, as there is an inherent tradeoff between profiling malware operating maliciously, such as gaining network access on a mobile device, and ensuring that both the malware remains within its sandbox and the malware profile remains accurate. Moreover, malware classifiers must be trained and evaluated in a repeatable and consistent manner with large-scale experimentation and automation infrastructure.

A. Challenge 1: Collecting Large Feature-Vector Datasets from Mobile Devices

Profiling 1,000s of applications on Android devices and emulators is commonly done in 2 ways: collecting crowdsourced data and traces from real user interaction, or emulating user interaction via fuzz testing. Crowdsourcing is advantageous to well-established malware detection methods due to the amount of real data acquired. However, crowdsourcing leads to privacy concerns, data limitations, and non-repeatability. Emulation is advantageous to experimental malware detection methods, as in our study.

Emulation is difficult due to the complex set up, management, and tear down process required. Automatic solutions are needed to install the APK to the device; start the application; simulate user interaction while collecting feature vectors; uninstall the application; keep track of the feature vectors; and train and test the classifiers. Our experiment applies these 7 steps to each application in our set of 1,738 applications, requiring over 12,000 total steps. If we had executed all steps in serial on one mobile device, it would require around seven days to complete execution.

While user input is being emulated on an application, feature vectors need to be collected. These will later be used to train the machine learning algorithms to detect feature usage indicative of malware. An effective malware classifier needs to continuously add new applications to the data set and profile them. As shown above, completing profiling in a timely manner is a key challenge to keeping malware classifiers up to date. Section III-A describes how our framework directly addresses this challenge via STREAM, a collection of automation approaches that address the aforementioned issues of APK installation, user input generation, feature vector collection, and malware classification.

B. Challenge 2: Accurately Collecting Malware Profiling Data on Mobile Devices

Android malware can cause privilege escalation, remote control, financial charges, and personal information stealing [5]. Sandboxing can prevent certain malware behavior, and therefore data collection must allow malware to both execute and control inherent safety issues. These issues can be local and affect a device, or remote and affect the network the device is attached to. This leads to a trade–off between malware execution and malware isolation. Locally, devices need to be constantly refreshed so malware won't be able to access data on the devices. Devices need to be exclusively designated for malware analysis and cannot contain any sensitive information.

For example, when profiling a malicious application, it is possible the malware can escape its sandbox and harm its surroundings. In our early experimentation, we improperly sandboxed device network traffic and leaked malicious traffic. The Virginia Tech IT Security Office detected this traffic and notified us of a malware infestation. For any cloud of devices allocated to run malware analysis, caution needs to be taken to prevent the devices from attempting to exploit computers connected to the cloud servers. We address the security concerns in Section III-B by discussing how we sandbox both applications and devices, only allow one malware to run per allocated device, and completely reformat a device between different tests.

III. A Framework for Distributed Application Profiling

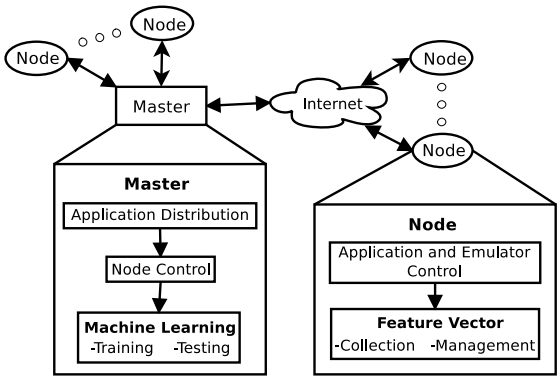

Fig. 1. High-level overview of STREAM distributed across multiple nodes.

STREAM, a System for automatically TRaining and Evaluating Android Malware classifiers, addresses the challenges discussed in Section II and provides an effective method of rapidly profiling malware and training machine learning classifiers. STREAM can run on a single server or distributed across a grid of remote servers. Figure 1 shows a high–level operational overview of STREAM. The *master* server distributes profiling jobs to the worker *nodes*.The node servers then distribute the jobs between devices or emulators in parallel. Inside each device or emulator, STREAM manages the applications, drives the feature vector collection, and manages classifier training and evaluation. The parallelization addresses the size of the problem domain and the STREAM framework provides a combination of accuracy and scalability.

A. STREAM Implementation Details

A primary challenge of effective mobile malware profiling, as described in Section II-A, is automating the process of infecting devices, generating realistic user input and test data to trigger malicious behavior, orchestrating the collection of this malware data from hundreds or thousands of devices, and continually training a large set of classifiers to measure malware classification performance. STREAM is an automation framework that addresses this challenge by infecting Android devices with app-based malware and simulating user-input to trigger malicious activity. STREAM manages the temporal synchronization of these activities, harvesting the output data, and training the malware classifiers. More broadly, STREAM is a framework for automating and synchronizing actions on Android emulators across tens of machines and thousands of emulators in tandem.

1) Test Platform: Infecting thousands of real physical devices is not feasible for most organizations. STREAM leverages the Android emulator as an option for its testing platform, enabling testing at scales impossible with physical devices.

2) Load Distribution Locally and Remotely: Running an emulator and collecting feature vectors in real-time is both time and memory intensive. Profiling an application for 5 minutes with a single emulator can require over 1GB of memory and 100% utilization on one core. STREAM can automatically manage running multiple emulators on both local and remote systems.

3) Device Preparation: STREAM prepares a device by configuring settings, creating an entire user persona, and installing any applications required for malware detection.

4) Device Infection: Drawing from a database of known malicious and benign applications obtained from the Malware Genome Project [5], VirusTotal Malware Intelligence Service [6], and Google Play, STREAM installs each application onto a device.

5) User Input Emulation: A critical component of multiple malware applications is user input. It is common for a malicious application, such as variants of AnserverBot or GGTracker, to inject its payload after a user approves an action. STREAM simulates this user input by invoking Android's Monkey, which sends a random stream of events to an application, while profiling applications. While Monkey generates acceptable data, it will not be as accurate as other, more sophisticated methods of generating input to test applications, such as crowdsourcing [7] and analysis-based fuzz testing [8]. Future work to STREAM will add a more robust and accurate method of generating user input.

6) Malware Execution Monitoring & Feature Vector Collection: While our experiments collected a set of feature vectors chosen for their use in prior work [9], future malware detection may require collection of different features. To support this, STREAM includes a modifiable Android application to give the researcher control over the contents of each feature vector. In our experimentation, we collect information about the battery, binder, memory, network, and permissions.

B. Enabling Accurate Malware Profiling with STREAM

It is necessary to balance unrestricted malware operation and security concerns arising from running malware. An obvious sandbox definition is the Android emulator, and therefore STREAM ensures application profile isolation by formatting an Android emulator after profiling. However, to avoid allowing malware testing to escape the sandbox, STREAM disables networking access. Networking has been shown to be a critical feature of some malware, such as botnets, and therefore future work in this area will explore allowing access to a controlled environment [5].

C. Results Management, Analysis, & Visualization.

Feature vector and classifier evaluation results are presented to users. This output is provided in ARFF format, which can be read into the Weka testing suite [10]. Weka provides capabilities to train and evaluate malware classifiers given feature vectors, and provides implemenatations of numerous popular machine learning algorithms. We use the random forest [11], naive Bayes [12], multilayer perceptron [13], Bayes net [14], logistic [15], and J48 [16] classifiers. These classifiers were chosen due to their use in similar research papers [9]. STREAM outputs the performance of the classifiers on all of the feature vectors and the feature vectors from each application individually. In addition to STREAM's output, Weka can be used manually to visualize the feature vectors from the ARFF files. For example, Figure 2 shows Weka charts for various feature vectors. Other features of Weka include preprocessing, clustering, and attribute selection algorithms on the data sets.

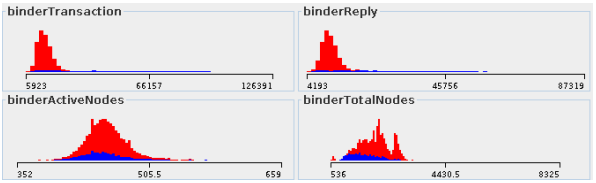

Fig. 2. Histograms created by Weka.

IV. Empirical Results

We present a study of the effectiveness of various malware classifiers trained and testing using STREAM. A data set of applications for training and testing were gathered from the Malware Genome Project [5], VirusTotal Malware Intelligence Service [6], and Google Play. The training set consists of 1738 applications (1330 malicious and 408 benign) and the testing set consists of 47 applications (24 benign and 23 malicious). There are 8 applications duplicated across both data sets, for a total of 1777 unique applications.

A. Experimental Platform

STREAM resides on the Android Tactical Application Assessment & Knowledge (ATAACK) Cloud [17], which is a hardware platform designed to provide a testbed for cloud–based analysis of mobile applications. The ATAACK cloud currently uses a 34 node cluster, with each cluster machine containing Dell PowerEdge M610 blade running CentOS 6.3.

Each node has 2 Intel Xeon 5645® processors with 12 cores each along with 36GB of DDR3 ECC memory.

In addition to this hardware platform, Samsung Nexus S devices are attached to the ATAACK Cloud to demonstrate the results of the framework when run on Android devices rather than emulators. These devices run a userdebug build of Android 4.0.4_r2.1, and the emulators are created with Google's SDK and Android 4.1.1.

B. Experiment 1: Malware Classifier Performance Across the Combined Data Set

Feature Vectors	
binderActiveTransactionComplete binderTotalTransactionComplete binderTotalTransactionDiff binderTotalTransactionCompleteDiff	
battIsCharging	battVoltage
battTemp	battLevel
battLevelDiff	binderTransaction
binderReply	binderAcquire
binderRelease	binderActiveNodes
binderTotalNodes	binderActiveRef
binderTotalRef	binderActiveDeath
binderTotalDeath	binderActiveTransaction
binderTotalTransaction	binderTotalNodesDiff
binderTotalRefDiff	binderTotalDeathDiff
cpuUser	cpuSystem
cpuIdle	cpuOther
memActive	memInactive
memMapped	memFreePages
memAnonPages	memFilePages
memDirtyPages	memWritebackPages
permissions	

TABLE I. FEATURE VECTORS COLLECTED WITH APPLICATION PROFILING

Term	Abbr.	Definition
True Positive	TP	Malicious app classified as malicious
True Negative	TN	Benign app classified as benign
False Negative	FN	Malicious app classified as benign
False Positive	FP	Benign app classified as malicious
True Positive Rate	TPR	$TP/(TP+FN)$
False Positive Rate	FPR	$FP/(FP+TN)$
Correctly Classified		Total number of correctly classified instances from the testing dataset

TABLE II. DEFINITIONS FOR THE PRESENTED RESULTS

We used STREAM to send 10,000 input events to each application in the data set and collect a feature vector every 5 seconds. We collected over thirty features in each vector, as shown in Table I. Feature vectors collected from the training set of applications were used to create classifiers, and then feature vectors from the testing set are used to evaluate the created malware classifiers. Classification rates from the testing set are based on the 47 testing applications used. Future work includes increasing the testing set size to increase confidence in these results.

Table II shows descriptions of the metrics used to evaluate classifiers. The overall results of training and testing six machine learning algorithms with STREAM are shown in Table III. There is a clear difference in correct classification percentage of the cross validation set (made up of applications used in training) versus the testing set (made up of applications never used in training). Feature vectors from the training set are classified quite well, typically over 85% correct, whereas new feature vectors from the testing set are often only classified 70% correctly. Classifier performance cannot be based on cross validation solely, as it is prone to inflated accuracy results.

When evaluating the classifiers on the testing set, which is a more realistic evaluation of classifier performance, Table III reveals Bayes' classifiers have a higher classification rate than other tested classifiers. Bayes net correctly classifies 81.25% of feature vectors, while the Naive Bayes classifier correctly classifies 78.91% of feature vectors. The Bayes' classifiers also reveal a distinction in the corresponding rates. The TPR of the Bayes' algorithms are 95.50% and 97.30%, while the TPR of the other algorithms range from 48.65% to 72.97%. The lower FPR rates of the other algorithms reveal the tradeoff. While the Bayes' algorithms tend to more accurately detect malware, the other algorithms tend to more accurately detect benign applications. Due to the high classification rate and high TPR of the Bayes net classifier, Section IV-C further analyzes the results from Bayes net on the specific applications from the testing set.

C. Experiment 2: Bayesian network Classifier Performance Across Individual Applications

This experiment uses the same sets of training and testing feature vectors from Section IV-B, but focuses on the performance of the Bayesian network classifier on individual feature vectors. While feature vectors from the testing set (i.e. unseen applications) were the focus, some feature vectors were from the training set and are marked as duplicated.

Table IV shows high detection rates for malicious applications. 86.36% of malicious applications were correctly classified with a 100.00% detection rate. Of these, all seven duplicated applications (e.g. included in the training set) all yielded 100.00% detection rates. Alternatively, only 52% of the benign applications were correctly classified with 100.00%. In addition, 17.39% of benign applications were classified with a 0.00% detection rate. These results would be unacceptable in a production program because the user would constantly receive false positive alerts claiming benign applications are malicious. Future work will explore ways of increasing detection rates by varying the features in the feature vectors, fuzz testing, and adding more applications to the data sets. Moreover, future work should directly examine potential sources of error, such as chosen features or emulated user behavior, and take steps to quantify the confidence in classifier analysis.

D. Analysis of Results

Our results in testing these classifiers show detection rates from 68.75% to 81.25%, representative of real data the classifiers will process in production. The evaluation time is comparable across on the testing set all classifiers, ranging from 0.261s to 0.728s. In addition to this, Table III shows the file sizes of the models can reasonably reside on a mobile device. The largest classifiers are Bayes net (2.7M) and multilayer perceptron (2.7M).

All of our results are based upon using the Android monkey user input tool, which lowers confidence that results are identical to real user input. Future work aims to incorporate advanced fuzz testing techniques from Mahmood et al. [8] to more accurately simulate user input. During our initial experimentation, our classifiers performed similar to random classification (50%) because few events with a delay were used to profile the applications to simulate real user input. We obtain

Classifier	Model Size	Cross Validation			Testing Set			
		Correctly Classified	TPR	FPR	Correctly Classified	TPR	FPR	Time
Random Forest	804K	94.53%	97.66%	14.85%	70.31%	66.67%	26.90%	0m 0.647s
Naive Bayes	17K	79.79%	87.85%	44.36%	78.91%	95.50%	33.79%	0m 0.261s
Multilayer Perceptron	2.7M	93.91%	97.27%	16.13%	70.31%	61.26%	22.76%	0m 0.728s
Bayes net	2.7M	86.23%	93.95%	36.88%	81.25%	97.30%	31.03%	0m 0.685s
Logistic	27K	89.52%	97.07%	33.08%	68.75%	48.65%	15.86%	0m 0.268s
J48	77K	93.43%	96.09%	14.55%	73.44%	72.97%	26.21%	0m 0.320s

TABLE III. RESULTS FROM MACHINE LEARNING CLASSIFIERS WITH 6,832 TRAINING FEATURE VECTORS EVALUATED WITH 10-FOLD CROSS VALIDATION AND AGAINST A TESTING SET OF 256 FEATURE VECTORS.

Benign Applications			
Name	Correct %	Name	Correct %
quiz.companies.game	66.67%	bbc.mobile.news.ww*	25.00%
battery.free	100.00%	mycalendarmobile	25.00%
android.reader	100.00%	imdb	100.00%
papajohns	0.00%	pinterest	25.00%
androidPlayer	100.00%	craigslistfree	0.00%
pregnancytracker	100.00%	hydra	100.00%
stylem.wallpapers	0.00%	bfs.ninjump	100.00%
templerun	100.00%	tumblr	0.00%
airpushdetector	100.00%	OMatic	0.00%
unveil	16.67%	box	100.00%
mahjong	100.00%	gtask	0.00%
songkick	100.00%		

Malicious Applications			
Name	Correct %	Name	Correct %
Beauty.Girl*	100.00%	Ipad2App	100.00%
XrayScanner	85.71%	ad.notify1	100.00%
mymovies	100.00%	gone60*	100.00%
CallOfDuty	94.12%	skyscanner	100.00%
DWBeta*	100.00%	antimosquitos	100.00%
android.installer	75.00%	sipphone*	100.00%
txthej*	100.00%	rommanager	100.00%
bowlingtime*	100.00%	paintpro*	100.00%
barcode	100.00%	zanti	100.00%
luckjesusblessings	100.00%	youLoveLiveWallpaper	100.00%
topGear	100.00%	fingerprint	100.00%

TABLE IV. PERFORMANCE OF THE BAYES NET CLASSIFIER ON INDIVIDUAL APPLICATIONS. *Duplicated applications

the current results by removing the delay between events and increasing the number of events from 1,500 to 10,000.

V. RELATED WORK

STREAM furthers the research in application profiling and dynamic malware analysis by providing a practical framework for existing methods to use. Dynamic malware analysis and application profiling to collect dynamic feature vectors are closely related. The malware detection and profiling methods are often presented together for this reason. The following provides an overview of current research in this area.

Dynamic malware analysis on the Android platform Most work in dynamic malware analysis studies kernel-level features. Much success has been reported from analyzing system calls on the Android platform [18], [19]. Network traffic is also a commonly analyzed feature for malware detection on the Android platform [20], [21].

Andromaly by Shabtai et al. [9] takes a different approach and analyzes higher level features obtainable from a user-space application, such as SMS and call usage; networking information; and power usage. Bente et al. [22] present similar features to Andromaly and add context and trust information. Kim et al. [23] monitor similar features on a mobile cloud infrastructure. Dini et al. [24] monitor similar features in addition to system calls.

Our work furthers the study of the feature vector collection methodology proposed by Shabtai et al. [9]. Most of the related work show advancements in technique, but not in scalability. Shabtai et al. use 40 benign applications and create 4 malicious applications for experimentation. Kim et al. use the GoldMiner malware variants to provide abnormal data. Dini et al. use 56 benign and 10 malicious applications for experimentation.

Application profiling for malware analysis Research in malware detection is correlated to application profiling to simulate realistic user input and trigger malicious activity. Crowdroid [7] collects behavioral–related data directly from users via croudsourcing and evaluates the data with a clustering algorithm. Mahmood et al. [8] combines fuzz testing with a framework to collect data in parallel on emulators. TaintDroid [25] presents a lightweight method to monitor the flow of sensitive data and detect anomalies. pBMDS [18] correlates user input to system calls at runtime. SmartDroid [26] combines dynamic analysis with static analysis to explore user interaction and malware trigger conditions.

STREAM implements the random fuzz testing provided by Android's monkey. Implementing a more sophisticated method of this testing is left for future work.

Android malware analysis frameworks and studies Similar frameworks to STREAM have been proposed. Bläsing et al. [27] present a sandbox to profile applications for static and dynamic analysis, but show no empirical results. Zhou et al. [28] evaluate 204,040 applications both statically (using Android permissions) and dynamically using Android API method calls, but do not consider application resource usage. Zheng et al. [26] analyzed 150,368 using a dynamic analysis approach based off of user-interface monitoring.

STREAM implements a dynamic analysis framework similar to the work by Bläsing and we present a study on 1,738 applications. The techniques used by Zhou and Zheng do not profile applications as discussed here. We are currently working on integrating a larger number of applications from Google Play into our datasets.

Android's Bouncer Android's Bouncer [29] provides automated scanning combined with static and dynamic analysis of applications submitted to Google Play. While it has decreased the number of malicious applications on Google Play, having an open platform for research in addition to this provides numerous benefits. 1) Researchers can share data and analysis techniques; 2) Antimalware vendors can benefit from the additional data; and 3) Antimalware studies based on dynamic analysis become more accessible to the research and corporate communities. Further, research by Oberheide and Miller [30] show numerous ways of circumventing Android's bouncer.

VI. Concluding Remarks & Lessons Learned

An important gap in research on dynamic mobile malware collection is a study of machine learning algorithm performance on mobile malware detection. A number of new malware detection approaches have been proposed, but without experimental baselines for comparison, determining what algorithms to start with when developing new techniques and how to validate those algorithms is hard. This paper presents results demonstrating the effectiveness of different machine learning classifiers in correctly classifying an arbitrary application as malware. Commonly used machine learning algorithms, such as multilayer perceptron and random forest, don't perform well. All of the classifiers, even Bayes net, which correctly classifies the highest percentage of feature vectors, had poor false positive rates, ranging from 15.86% to 33.79% in the testing set.

From our experiments with malware classifiers, we learned the following important lessons: 1) results from cross validation, which are heavily used in the literature due to a lack of malware samples, are not consistent with results from real testing and over-estimate performance 2) Machine-invariant features should be explored in future work, as feature vectors can vary drastically from one platform to the next. 3) While our malware set was large relative to prior studies, future work aims to download a large database of applications from Google Play and run a much larger benchmark.

This work is available under the `IWCMC-2013` tag at https://github.com/VT-Magnum-Research/antimalware.

References

[1] "Android Marks Fourth Anniversary Since Launch with 75.0% Market Share in Third Quarter, According to IDC," URL http://www.idc.com/getdoc.jsp?containerId=prUS23771812, 2012.

[2] "Google Says 700,000 Applications Available for Android," URL http://www.businessweek.com/news/2012-10-29/google-says-700-000-applications-available-for-android-devices, 2012.

[3] N. Idika and A. P. Mathur, "A survey of malware detection techniques," *Purdue University*, p. 48, 2007.

[4] "Malware Detection Methods," URL http://www.avg.com/us-en/avg-software-technology.

[5] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," in *Security and Privacy (SP), 2012 IEEE Symposium on*, May 2012, pp. 95 –109.

[6] "VirusTotal Malware Intelligence Services," URL https://secure.vt-mis.com/vtmis/.

[7] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for android," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, ser. SPSM '11. New York, NY, USA: ACM, 2011, pp. 15–26. [Online]. Available: http://doi.acm.org/10.1145/2046614.2046619

[8] R. Mahmood, N. Esfahani, T. Kacem, N. Mirzaei, S. Malek, and A. Stavrou, "A whitebox approach for automated security testing of Android applications on the cloud," in *Automation of Software Test (AST), 2012 7th International Workshop on*, June 2012, pp. 22 –28.

[9] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, ""Andromaly": a behavioral malware detection framework for android devices," *Journal of Intelligent Information Systems*, vol. 38, pp. 161–190, 2012, 10.1007/s10844-010-0148-x. [Online]. Available: http://dx.doi.org/10.1007/s10844-010-0148-x

[10] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, Nov. 2009. [Online]. Available: http://doi.acm.org/10.1145/1656274.1656278

[11] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[12] P. Domingos and M. Pazzani, "On the optimality of the simple bayesian classifier under zero-one loss," *Machine learning*, vol. 29, no. 2, pp. 103–130, 1997.

[13] F. Rosenblatt, "Principles of neurodynamics. perceptrons and the theory of brain mechanisms," DTIC Document, Tech. Rep., 1961.

[14] R. R. Bouckaert, "Bayesian network classifiers in weka for version 3-5-6."

[15] D. Hosmer and S. Lemeshow, "Logistic regression for matched case-control studies," *Applied logistic regression*, vol. 2, pp. 223–259, 1989.

[16] J. R. Quinlan, *C4. 5: programs for machine learning*. Morgan kaufmann, 1993, vol. 1.

[17] H. Turner, J. White, J. Reed, J. Galindo, A. Porter, M. Marathe, A. Vullikanti, and A. Gokhale, "Building a cloud-based mobile application testbed," *Software Testing in the Cloud: Perspectives on an Emerging Discipline*, Nov 2012.

[18] L. Xie, X. Zhang, J. Seifert, and S. Zhu, "pbmds: a behavior-based malware detection system for cellphone devices," in *Proceedings of the third ACM conference on Wireless network security*. ACM, 2010, pp. 37–48.

[19] T. Isohara, K. Takemori, and A. Kubota, "Kernel-based behavior analysis for android malware detection," in *Computational Intelligence and Security (CIS), 2011 Seventh International Conference on*. IEEE, 2011, pp. 1011–1015.

[20] T. Wei, C. Mao, A. Jeng, H. Lee, H. Wang, and D. Wu, "Android malware detection via a latent network behavior analysis," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*. IEEE, 2012, pp. 1251–1258.

[21] J. Cucurull, S. Nadjm-Tehrani, and M. Raciti, "Modular anomaly detection for smartphone ad hoc communication," *Information Security Technology for Applications*, pp. 65–81, 2012.

[22] I. Bente, B. Hellmann, J. Vieweg, J. von Helden, and G. Dreo, "Tcads: Trustworthy, context-related anomaly detection for smartphones," in *Network-Based Information Systems (NBiS), 2012 15th International Conference on*. IEEE, 2012, pp. 247–254.

[23] T. Kim, Y. Choi, S. Han, J. Chung, J. Hyun, J. Li, and J. Hong, "Monitoring and detecting abnormal behavior in mobile cloud infrastructure," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*. IEEE, 2012, pp. 1303–1310.

[24] G. Dini, F. Martinelli, A. Saracino, and D. Sgandurra, "Madam: a multi-level anomaly detector for android malware."

[25] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924943.1924971

[26] C. Zheng, S. Zhu, S. Dai, G. Gu, X. Gong, X. Han, and W. Zou, "Smartdroid: an automatic system for revealing ui-based trigger conditions in android applications," in *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices*. ACM, 2012, pp. 93–104.

[27] T. Blasing, L. Batyuk, A.-D. Schmidt, S. A. Camtepe, and S. Albayrak, "An android application sandbox system for suspicious software detection," in *Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on*. IEEE, 2010, pp. 55–62.

[28] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets," in *Proc. of the 19th Annual Network and Distributed System Security Symposium (NDSS)*, 2012.

[29] "Android and Security," URL http://googlemobile.blogspot.com/2012/02/android-and-security.html, 2012.

[30] J. Oberheide and C. Miller, "Dissecting the android bouncer." SummerCon 2012, 2012.