POSTER: Semi-supervised Classification for Dynamic Android Malware Detection

Li Chen, Mingwei Zhang, Chih-yuan Yang, Ravi Sahita
Security and Privacy Lab, Intel Labs, Hillsboro, OR

ABSTRACT

Manually labeling the large number of samples of Android APKs into benign or different malicious families requires tremendous human effort, while it is comparably easy and cheap to obtain a large amount of unlabeled APKs from various sources. Moreover, the fast-paced evolution of Android malware continuously generates derivative and new malware families. These families often contain new signatures, which can escape detection that uses static analysis. These practical challenges can also cause classical supervised machine learning algorithms to degrade in performance.

We propose a framework that uses model-based semi-supervised (MBSS) classification scheme built using dynamic Android API call logs. The semi-supervised approach efficiently uses the labeled and unlabeled APKs to estimate a finite mixture model of Gaussian distributions via conditional expectation-maximization and efficiently detects malware during out-of-sample testing. We compare MBSS with the popular malware detection classifiers such as support vector machine (SVM), k-nearest neighbor (kNN) and linear discriminant analysis (LDA). Under the ideal classification setting, MBSS has competitive performance with 98% accuracy and very low false positive rate for in-sample classification. For out-of-sample testing, the out-of-sample test data exhibit similar behavior of retrieving phone information and sending to the network, compared with in-sample training set. When this similarity is strong, MBSS and SVM with linear kernel maintain 90% detection rate while kNN and LDA suffer great performance degradation. When this similarity is slightly weaker, all classifiers degrade in performance, but MBSS still performs significantly better than other classifiers.

KEYWORDS

Android dynamic malware analysis, semi-supervised machine learning, robust machine learning

1 MOTIVATION

The rampant evolution of Android malware makes it more sophisticated to avoid detection and more difficult to classify using commonly-used machine learning algorithms. Human effort of labeling the malware as malicious or benign cannot keep up with the pace of the voluminous generation of Android malware, resulting in an imbalance of much more unlabeled data than labeled data. Even VirusTotal could take a long time to finalize the prediction of large amount of unlabeled data. Further, this can cause supervised learning algorithms to degrade in performance.

Furthermore, due to this imbalance of unlabeled and labeled data, the distribution observed from the labeled data can be different from the actual data distribution. This is seen in malware detection, as study suggests that malware family exhibit polymorphic behaviors. Translated into machine learning, it implies that at testing phase, the test data can be similarly distributed as the training data, but is not identically distributed as the training data. Classical supervised machine learning algorithms can suffer performance degradation when tested on samples that do not distribute identically as the training data.

2 PROPOSED FRAMEWORK

To address the above challenges, we propose a framework of utilizing model-based semi-supervised (MBSS) classification on the dynamic behavior data for Android malware detection. We focus on detecting malicious behavior at runtime by using dynamic behavior data. The main advantage of semi-supervised classification is the strong robustness in performance for out-of-sample testing. The model-based semi-supervised classification uses both labeled and unlabeled data to estimate the parameters, since unlabeled data usually carries valuable information on the model fitting procedure.

Specifically, we use mixture modeling to achieve the classification task. Mixture modeling is a type of machine learning technique, which assumes that every component of the mixture represents a given set of observations in the entire collection of data. Gaussian mixture modeling is the most popularly used applied and studied technique.

We run the Android applications in our emulator infrastructure and harvest the API calls at runtime. Our framework efficiently uses the labeled and unlabeled behavior data to estimate a set of finite mixture models of Gaussian distributions via conditional expectation-maximization, and uses the Bayesian information criterion for model selection. We compare MBSS with the popular malware detection classifiers such as SVM, kNN and LDA. We demonstrate that MBSS has competitive performance for in-sample classification, and maintains strong robustness when applied for out-of-sample testing. We consider semi-supervised learning on dynamic Android behavior data a practical and valuable addition to Android malware detection.

3 DATA GENERATION AND PREPARATION

Malicious and benign APKs were downloaded from VirusTotal and Google Play respectively. Both samples were uploaded to our emulator infrastructure for execution. Our Android emulator (AE) runs in a emulator machine. Each machine may contain multiple emulators. The downloaded APKs were dispatched by the scheduler and installed to new emulator instances for execution. Since most of

CCS '17, October 30-November 3, 2017, Dallas, TX, USA
© 2017 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-4946-8/17/10.
https://doi.org/10.1145/3133956.3138838

Android applications are UI-based, merely launching the application may be insufficient to expose its behaviors. An automation tool is developed to provide simulated human interactions, such as clicks, and sensor events, such as GPS location. In addition, this tool navigates the UI automatically without human intervention. We harvest applications behavior logs through Android Debug Bridge (ADB) [1] and aggregate them into the disk.

We capture dynamic behavior of each android application by executing it in our emulator. The instrumentation of Android runtime was implemented by Xposed [6] in a similar environment like CuckooDroid[5]. The dynamic log consists of a time sequence of API calls made by an application to the Android runtime. Since current Android runtime export more than 50K APIs, for efficiency, we carefully select 160 API calls that are critical to change Android system state such as sending short messages, accessing website, reading contact information, etc. Our selection of Android API function comes from the union of function set selected by three well-known open source projects: AndroidEagleEye [3], Droidmon [4], and Droidbox [2].

As a result, our data consists of the dynamic traces of the APKs as samples, and the feature space of our data set is the collection of n-gram of APIs. That is, suppose the unique number of n-concatenated API calls is d, then a sample APK is represented by $x = (0, 0, .., 1, .., 1, .., 0) \in \{0, 1\}^d$, where 1 denotes the existence of an n-API call sequence and 0 otherwise.

4 MODEL-BASED SEMI-SUPERVISED CLASSIFICATION

In the model-based semi-supervised approach, the data x is assumed to be distributed from a mixture density $f(x) = \sum_{k=1}^{K} \pi_k f_k(x)$, where $f_k(\cdot)$ is the density from the k-th group and π_k is the probability that an observation belongs to group k. Each component is Gaussian, which is characterized by its mean μ_k and covariance matrix Σ_k. The probability density function for the k-th component is thus

$$f_k = f_k(x, \mu_k, \sigma_k) = \frac{\exp(-\frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1}(x - \mu_k))}{\sqrt{\det(2\pi\Sigma_k)}}. \tag{1}$$

The mean $\mu = (\mu_1, ..., \mu_K)$, covariance matrix $\Sigma = (\Sigma_1, ..., \Sigma_K)$ and the population distribution $\pi = (\pi_1, ..., \pi_K)$ are the parameters to be estimated from the mixture models.

Denote $\theta := (\mu, \Sigma)$ as all the parameters in the Gaussian components. Hence we want to estimate π and θ. Denote $\mathcal{X}_n = \{X_1, ..., X_n\}$ as the training data and $\mathcal{Y}_n = \{Y_1, ..., Y_n\}$ as the training labels, where for the i-th observation, denote $Y_{ik} = 1$ if the observation comes from group k and 0 otherwise. Denote the unknown labels of the unlabeled data as $\mathcal{Y}_M = \{Y_{n+1}, ..., Y_{n+m}\}$.

In our framework, we apply model-based semi-supervised classification using both labeled and unlabeled Android behavioral data to develop the classification decision for the unlabeled data. The likelihood of the complete-data consisting of labeled and unlabeled data is

$$\begin{aligned} &L_C(\pi, \theta | \mathcal{X}_n, \mathcal{Y}_n, \mathcal{X}_m, \mathcal{Y}_m) \\ &= \Pi_{i=1}^{n} \Pi_{k=1}^{K} [\pi_k f(x_i|\theta_k)^{Y_{ik}}] \Pi_{j=n+1}^{n+m} \Pi_{k=1}^{K} [\pi_k f(X_j|\theta_k)]^{Y_{jk}}. \end{aligned} \tag{2}$$

Essentially we treat the data with unknown labels as missing data to include them in the complete likelihood.

To estimate the unknown parameters and maximize the log-likelihood of the complete data, we use the conditional expectation-maximization (CEM) algorithm [8]. It is used to solve the likelihood maximization on the complete data, and consists of four main steps:

- **Initialization**;
- **Calculate expectation**;
- **Maximize the expectation**;
- **Stop until convergence**.

Parameter estimation via CEM results in multiple models depending on the Gaussian shapes. We proceed with model selection using the Bayesian information criterion, given by

$$\text{BIC}(M_i) = 2 \log L_C - \ln(n + m)l, \tag{3}$$

where $\log L_C$ is the maximum likelihood of the complete data, $n + m$ is the number of obeservations, and l is the number of parameters.

5 RESULTS

We experiment using the dynamic logs obtained from our Android emulator. We compare MBSS with some of the most popular malware classifiers: SVM, kNN and LDA. We conduct two categories of experiments: in-sample validation and out-of-sample validation. In-sample validation is the typical classification setting where a dataset is split into training and test set. A classifier is trained on the training set and expected to perform well when tested on the test set. Cross validation is usually employed to assess the performance of a classifier. In-sample classification provides the ideal classification scenario, where test data distribution is the same as the training data distribution.

Our next set of experiments focuses on out-of-sample testing. An out-of-sample experiment uses the classifier trained and validated on the in-sample data, and predicts the labels of incoming unlabeled data. The vast majority of the unlabeled data are not guaranteed to follow the same distribution as the in-sample training data. This is a challenge for machine learning algorithms used for practical applications. We consider a classifier robust and practical when it can still achieve reasonably well performance for out-of-sample classification. All the APKs for our experiments are retrieved from VirusTotal, and the experiments are conducted in R [9], [10], [7].

5.1 In-sample validation

We first demonstrate that for in-sample classification, MBSS has competitive performance when compared with SVM with radial kernel, SVM with linear kernel, 3NN and LDA.

As in-sample dataset, we obtain 55994 Android APK dynamic logs from our emulator with 24217 benign Android APKs and 31777 malicious APKs. The in-sample malicious behaviors include stealing location, device ID, MAC information, dynamic code loading behavior, and sending the information to outside network. The label distribution is (43%, 57%) and the chance accuracy (classification accuracy when randomly guessing) is 0.57.

We conduct 10-fold cross validation, report the accuracy mean by averaging the accuracies and calculate the standard deviation of the accuracies across all the 10 folds, and report the similar metrics for false positive rates. As indicated in Table 1, all the classifiers have

Table 1: Classification performance comparison for all five classifiers. All classifiers have competitive classification performance for in-sample testing. For OOS1, MBSS and SVM with linear kernel achieve the highest detection rate. For OOS2, MBSS performs significantly better than all other classifiers. Due to too many ties for 3NN, we do not report its result here.

Classifier	Mean ACC	Sd ACC	Mean FP	Sd FP	DR for OOS1	DR for OOS2
MBSS	97.6%	0.002	3%	0.004	90.0%	55.3%
SVM (radial)	98.8%	0.002	1.8%	0.003	0	0.06%
SVM (linear)	98.6%	0.001	2%	0.003	90.8%	35.4%
3NN	97.9%	0.001	3%	0.004	68.4%	NA
LDA	90.0%	0.003	6%	0.003	9.4%	33.8%

competitive performance. Under this ideal classification scenario, SVM with radial and linear kernels demonstrate the best performance with accuracies at 98.8% and 98.6% respectively and false positive rates both at 2%, 3NN and MBSS have similar performance of accuracy at 97.6% and 97.9% respectively and false positive rate at 3%, while LDA shows lesser performance with accuracy at 90% and false positive at 6%. The receiver operating curve (ROC) of MBSS in the 10-th fold classification and the area under the curve is 0.99.

5.2 Out-of-sample classification

Here we report the detection rate (DR), which is defined by the number of correctly classified malware APKs divided by the total number of out-of-sample test data. After validating that these classifiers have high accuracy and low false positive in Section 5.1, we use them to test on incoming samples. Under this practical and realistic scenario, the test samples do not follow very similar distribution as the training samples. In this case, the classifiers with high accuracy degrade, sometimes even significantly, for out-of-sample testing.

5.2.1 OOS1: Out-of-sample with malicious similarity to in-sample data. We first apply the five classifiers on a dataset of 12185 malicious APKs and report the detection rate. These out-of-sample APKs exhibit similar malicious behaviors of intercepting and sending messages without the user's consent as in the training set.

To visualize this similarity, we conduct principal component analysis (PCA) and inspect the scatter plots of the first four principal components (PC). (See poster for details)

The sixth column in Table 1 demonstrates the detection rates of the five classifiers. Both SVM with linear kernel and MBSS have detection rate of 0.9, while the performance of 3NN is 0.68 and LDA has the lowest detection rate of 0.1. A dramatic performance degradation is seen for SVM with radial kernel, which went from the best in-sample performing classifier to the worst classifier with detection rate near 0. The significant degradation of LDA is due to its highly parametric nature. In this case, SVM with linear kernel and MBSS maintain relatively stable detection rate.

Next, we examine the detection rate as we vary the test size. We apply the classifiers on the randomly selected $\{0.1\%, 1\%, 20\%, 50\%, 90\%\}$ of the test data with independent Monte Carlo replications at $\{50, 30, 20, 10, 5, 1\}$ respectively. We also observe the superior performance of MBSS at 0.9, compared with other classifiers.

5.2.2 OOS2: Out-of-sample with dissimilar distribution to in-sample data. Our next out-of-sample experiment applies the five classifiers onto a dataset of 11986 malicious APKs, whose malicious behaviors primary include stealing private information, sending it to the Internet through commodity command and control (C&C) server, but do not include dynamic code loading behavior. This slight similarity in malicious behavior can been seen in the data distribution as reflected by the principal components achieved from principal component analysis.

The last right column in Table 1 demonstrates the detection performance of the five classifiers. All classifiers degrade in performance. However SVM with linear kernel degrades in performance significantly. The result of 3NN is omitted, because it fails due to too many ties. MBSS performs significantly better than the other classifiers.

We vary the test size to examine the out-of-sample classification performance. As we vary the test size, we apply the classifiers on the randomly selected $\{0.1\%, 1\%, 20\%, 50\%, 90\%\}$ of the test data with independent Monte Carlo replications at $\{50, 30, 20, 10, 5, 1\}$ respectively. We also observe the robust performance of MBSS compared to the great degradation by other classifiers.

6 CONCLUSION

In this poster, we demonstrate the effectiveness of using model-based semi-supervised learning (MBSS) approach on dynamic Android behavior data for Android malware detection. We show that for in-sample testing, MBSS has competitive accuracy and false positive rate compared with the most popular malware classifiers. For out-of-sample testing, MBSS produces significantly higher detection rate compared with the other classifiers in consideration. We are optimistic that the framework of semi-supervised learning for dynamic analysis is valuable and practical for anti-malware research.

REFERENCES

[1] 2008. Android Debug Bridge. https://developer.android.com/studio/command-line/adb.html. (2008).
[2] 2012. Droidbox. https://github.com/pjlantz/droidbox. (2012).
[3] 2014. MindMac/AndroidEagleEye. https://github.com/MindMac/AndroidEagleEye. (2014).
[4] 2015. Droidmon. https://github.com/idanr1986/droidmon. (2015).
[5] 2016. CuckooDroid - Automated Android Malware Analysis. https://github.com/idanr1986/cuckoo-droid. (2016).
[6] 2016. rovo89/Xposed. https://github.com/rovo89/xposed. (2016).
[7] Chris Fraley, Adrian E. Raftery, Thomas Brendan Murphy, and Luca Scrucca. 2012. *mclust Version 4 for R: Normal Mixture Modeling for Model-Based Clustering, Classification, and Density Estimation.*
[8] Tony Jebara and Alex Pentland. 1998. Maximum conditional likelihood via bound maximization and the CEM algorithm. In *Proceedings of the 11th International Conference on Neural Information Processing Systems.* MIT Press, 494–500.
[9] R Core Team. 2016. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria.
[10] Niamh Russell, Laura Cribbin, and Thomas Brendan Murphy. 2014. *upclass: Updated Classification Methods using Unlabeled Data.* R package version 2.0.