An Android Behavior-Based Malware Detection Method using Machine Learning

Wei-Ling Chang and Hung-Min Sun
Dept. of Computer Science
National Tsing Hua University
Hsinchu, Taiwan
hmsun@cs.nthu.edu.tw

Wei Wu
Fujian Provincial Key Laboratory of Network Security and Cryptology, School of Mathematics and Computer Science
Fujian Normal University
Fuzhou, Fujian, China
weiwu@fjnu.edu.cn

***Abstract*—In this paper, we propose An Android Behavior-Based Malware Detection Method using Machine Learning. We improve an Android application sandbox, Droidbox, by inserting a view-identification automatic trigger program which can click mobile applications in the meaningful order. Taking advantage of Droidbox result, we collect the behavior such as network activities, file read/write and permission as the feature data and use different machine learning algorithms to classify malware and evaluate the performance. We use a large number of malware and normal application samples to prove that our method has high accuracy.**

***Keywords*—Android Malware, Behavior-based Analysis, Machine learning, Software Security**

I. INTRODUCTION

Android OS has become the leader of the global smart phone market. One of the superiority of Android is its open source licenses, and it has designed its software developer kit (SDK) to work across as many platforms as possible[1]. The open source Android platform is more flexible than iOS and allows developers to take full advantage of the mobile operation system. The popularity of Android system attracts many developers to develop not only useful and creative applications, but also some malicious software.They insert malware on Android Google play or other third party platform, pretending to be normal applications. Malware on Android can not merely steal users privacy information such as device information, phone number, IMEI, contact list, credit card number, bank account numbers and even make the device be a C&C server botnet. The rest of this paper is organized as follows. In Section II, we will introduce some malware analysis method in the past. In Section II, we review Patrik's scheme and the improved method. In Section III, we describe our system architecture and the implementation of our framework. In Section IV, we discuss the result. In Section V, we talk about our conclusion.

II. RELATED WORKS

In section II-A, we first briefly introduce Patrik's protocol. The weakness of this sand box is shown in sectionII-B. Finally, we will discuss some improved methods.

A. Review of Patrik's protocol

In 2011, Patrik[2] presented Droidbox, an Android application sandbox[3-4] for dynamic analysis. It was the first publicly available tool for dynamic analysis on mobile platform. The process of Droidbox includes three parts: static pre-check, monkeyrunner, and logcat filter, which is shown in Figure 1.

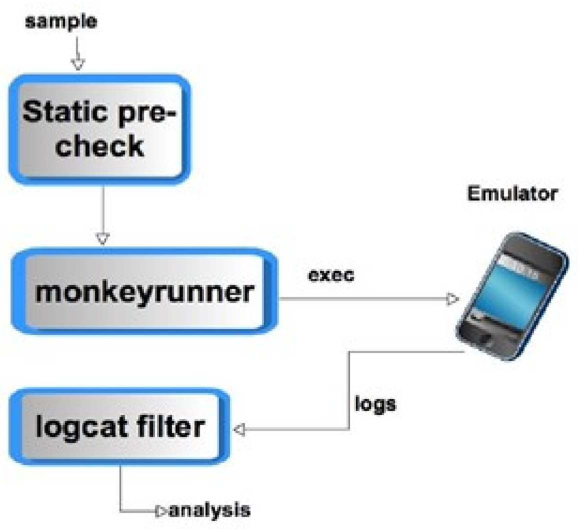


Fig. 1. Droidbox Process

1) Static pre-check

Before executing application, static pre-check unzip a package and parse the Manifest. More precisely, in this part, system extracts application main activity, Java package and some information relevant for analysis such as permission and registers Intent receivers from Manifest file. Java package and main activity are used as the parameters of the next process "monkeyrunner script". For each registered receiver, this process also parses the action tag within these component sintent-filtertags. This tag specifies what kind of the broadcast message the component receives.

2) Monkeyrunner script

Monkeyrunner script automatically installs package and launches this application main activity or specific service. Monkeyrunner needs two parameters: application package name and main activity name which has already been extracted in static pre-check step.

3) Logcat filter

While executing, logcat filter uses SDK tool logcat to intercept logs which are broadcasted from the emulator. Droidbox pipelines the logcat output to a Python script that is read from stdin. They also increase the maximum length of one log from four thousand characters to twenty thousand characters. This change has an effect to solve the problem which kernel will truncate logs that are too long. Logcat filter collects each type of log, stored by the timestamp relative to the starting time of the analysis for when the log was collected.

B. Weakness of Patrik's protocol

Droidbox is the first open-source mobile sandbox for dynamic analysis. Undeniably, it really helps developer to do detection on mobile platform more convenient than before. However, the latest version of Droidbox is released in 2013. It was three years before. In our opinion, now it is inflexible for analyst to detect malware on mobile platform by only using Droidbox. The disadvantages are as follows.

1) No automatic stress-test

Droidbox, by default, does not provide automatic stress-test to simulate user interaction or analyst needs to click the screen of emulator manually. If we need to analyze large number of malicious software, it is time consuming and very inefficient.

2) No decision model

As we mentioned before, the information of the application's action during runtime is listed in analysis result. However, it is not easy to determine whether this application is a malware just by these collected behvior.

III. System Framework

Our system mainly consists of four components: Preprocessing, Data Monitor, Decision Model, Database, as shown in Figure 2. Next, I will introduce each part in detail.

A. Preprocessing

First, we need to get the target application package name and main activity name from the file "Manifest". Next, delete the original signature of the target application. The reason is that the trigger program needs the same signature as the target application. Third, create one Android key for signature (do this step only the first time). Then, use the same key to sign the trigger program and target application and install both in the Android emulator.

B. Data Monitor :

In Figure 3, we show our detail framework of Trigger Process and Data Monitor. Once Droidbox finishes static pre-check, InstrumentationTestRunner will automatically execute Robotium trigger package. The target application is automatically started by the trigger program, and do the defined events in Android emulator.

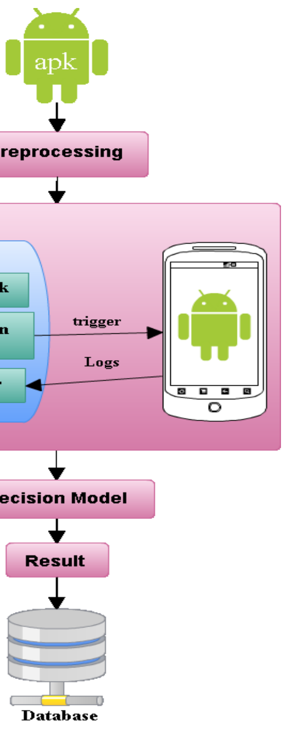


Fig. 2. System Framework

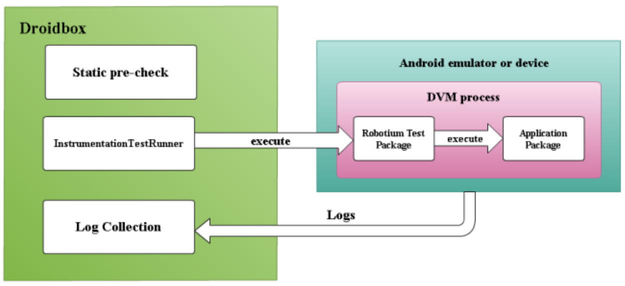


Fig. 3. Tigger Process

In this subsection, we will talk about our framework of Robotium trigger program in detail. Figure 4 illustrates the flow chart of our trigger process. Next we will explore each procedure.

1) Get View and filter: View is the base class for widgets, which are used to create interactive UI components. (ListView, Buttons, TextView, etc.) In order to do trigger progress, we need to get the views of current UI window. Robotium is friendly for developers. It provides solo.getCurrentViews(), a function which returns an ArrayList of the views currently displayed in the focused Activity or Dialogue. Besides, we need to delete these from the current view-arraylist; otherwise, the trigger program will produce an error and terminate the trigger process. In addition, we will filter the clicked views on this activity and only click the views which are not clicked before.

2) Rule: Package android.widget is a view component used to interact with users. This package contains UI elements to use on your application screen and provides many UI class such as Button, CheckBox, EditText, ListView, etc. Each widget provides different function. Basically, we divide widgets into two groups, action widget and parameter widget

• **action widget:** The widgets provide action event such as button, image button.

• **parameter widget:** The widgets need input parameter such as edittext, spinner, etc.

We have got the visible and clickable views on current UI window from previous procedure. The way we think is that the clicking order must be similar to normal user's habit. The following are our rules of trigger program.

• **Rule one**: Parameter widget generally should be triggered before action widget. In some cases, action widget will do some operations on the input from parameter widget. If action widget is triggered before parameter widget, there may be some errors on program execution flow.

• **Rule two**: We will distinguish which action widget on current UI window has higher priority when there are more than one action widget on current UI window.

• **Rule three**: Parameter widgets must be full of suitable value to lead them to the right state. Our method is to look for keyword from widget ID, input type, hint associated with edittext or the text label next to the edittext.

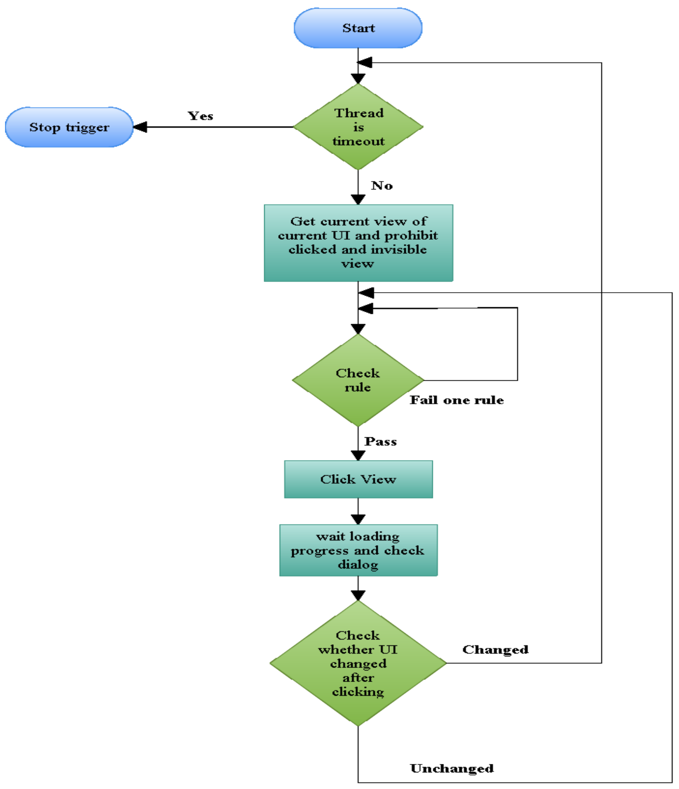

Start
Stop trigger
Yes
Thread is timeout
No
Get current view of current UI and prohibit clicked and invisible view
Check rule
Fail one rule
Pass
Click View
wait loading progress and check dialog
Check whether UI changed after clicking
Changed
Unchanged

Fig. 4. Flow chart of our trigger process

• **Rule four**: Widgets in scrollable container should be triggered before scrolling since scrollable container will hide some widgets inside after scrolling.

• **Rule five**: If all views in current UI window are clicked already and the current activity is not the beginning launch activity, then perform function "go back" to the previous activity.

3) Examination: After passing the rule, we click the specified view. Each press operation probably results another different UI window displayed. Therefore, we will do the following examination after each press operation.

•**Wait loading**: While some applications are in the loading phase, a progress bar is displayed on the mobile screen to imply users the current program schedule. We can not do any operation until progress bar finished.

•**Check dialogue**: If there is a dialogue emerged on the mobile screen, our trigger program will get the visible view on this dialogue and click the prioritized button or view.

•**Check whether UI (activity) is changed**: We match the activity before click with the activity after click. If they are the same, we do the next rule examination. Otherwise, restart all testing progress.

4) Termination point: We set a fix trigger period. We create a thread to control when to terminate the trigger process. The trigger program will be terminated when thread is time out. Based on observation and efficiency, we consider that 5 minutes is enough for collecting feature data.

C. Decision Model:

Decision model is the main function of our framework. We build our decision model using machine learning method. In this subsection, we will introduce the behavior we collect during execution time and machine learning algorithms we used. We also discuss our machine learning process in detail.

1) Feature data: In this paper, our feature data is mainly divided into two parts: dynamic behavior and permission.

• **Dynamic behavior**

During the execution time, we collect dynamic behavior as follows:

1. File read/write

2. Crypto usage: Detect the used cryptography functions and what key is used when encrypting and decrypting data.

3. Network open/close, receive/send: Record the network activities, opened/closed connection and read/write operation.

4. Started services and DexClassLoader: Log the started service and any operations of loading external DEX function.

5. Broadcast receiver: Record the broadcast receiver declared in this APK.

6. Information leak: Detect leaks of sensitive information on the phone including email, password, contacts, SMS data, IMEI, GPS coordinates, phone number and so on.

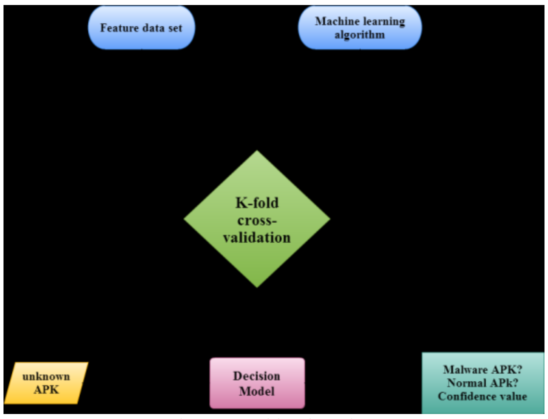
7. Phone call and sent SMS: Logs SMS message and received phone number.

8. Enforced permission: Record the permission provided to other APK.

• **Permission**

Collect all sample APK permissions to assist detection.

2) Machine learning process: We show our machine learning process in Figure 6. With feature data set we collected and different machine learning algorithms, we build a suitable decision model. During the building process, we use K-fold cross validation to evaluate the model. We can use our decision model to determine the unknown APK to be malware or normal APK and show the confidence value. Confidence value is the class probability distribution for an unknown APK.


Feature data set
Machine learning algorithm
K-fold cross-validation
unknown APK
Decision Model
Malware APK? Normal APk? Confidence value
Fig.6 . Machine learning process

IV. Discussion

A. Evaluation result

In this subsection, we will show our classification result of feature data set, permission and dynamic behavior, by using different machine learning algorithms. In table I, it is obvious that True Positive Rate and Accuracy is up to 97 % and False Positive Rate is lower than 10%. The result shows True Positive Rate of Robotium is better than True Positive Rate of Monkey and False Positive Rate of Robotium is lower than False Positive Rate of Monkey.

TABLE I. Evaluation indicators value of all feature data

Monkey				Robotium		
TPR	FPR	Accuracy	Algorithm	Accuracy	FPR	TPR
0.966	0.049	96.1%	RandomForest	97%	0.036	0.971
0.951	0.09	93.6%	J48	95%	0.08	0.97
0.96	0.079	94.6%	LMT	96.1%	0.064	0.978
0.972	0.084	95.2%	RandomForest	96.2%	0.053	0.973
0.914	0.133	89.7%	LogitBoost	93%	0.095	0.945
0.944	0.087	93.3%	Bagging	95.1%	0.076	0.969
0.971	0.133	93.3%	IBK(KNN)	95.3%	0.082	0.977
0.972	0.126	93.6%	Ksatr	95.8%	0.072	0.978
0.957	0.085	94.17%	PART	96.1%	0.057	0.974
0.833	0.063	87.1%	BayesNet	90.7%	0.057	0.881

B. Comparison

Tsai[5] proposed an Android machine learning malware detection. His feature data includes permission, network packet leakage and system call n-gram. He also triggered android application based on Robotium Framework. Table II presents each evaluation indicator result of our system and Tsai's. In this paper, there are three points differnet with Tsai. First, our trigger program support application webview click. Second, after trigger procress, we store the collected behavior as a report. Third, Tsai does not implement the process of using model to detect unknown APK. The result shows that our Robotium program is better than Tsai. With each algorithm, True Positive Rate and Accuracy of our protocol are higher than Tsais and False Positive Rate is lower. The difference in accuracy between Tsai and our method is up to 8.

TABLE II. Comparison of our method and Tsai[5]

Tsai[5]				Robotium		
TPR	FPR	Accuracy	Algorithm	Accuracy	FPR	TPR
0.967	0.062	95.5%	RandomForest	97%	0.036	0.971
0.946	0.085	93.4%	J48	95%	0.08	0.97
0.907	0.153	88.4%	RandomCommittee	96.2%	0.053	0.973
0.937	0.095	92.5%	Bagging	95.1%	0.076	0.969
0.968	0.099	94.2%	IBK(KNN)	95.3%	0.082	0.977

V. Conclusion

We proposed a Robotium program in an Android sandbox which can trigger Android application automatically and monitor behavior. A Robotium program is a UI-identification automatic trigger program which can click mobile applications in meaningful order. More importantly, we do large scale experiments. We build a decision mode lusing behavior we collected and RandomForest algorithm. It can determine whether unknown application is a malware and show its confidence value. Our evaluation indicators show permission and dynamic behavior we collected such as network activtiy, file read/wirte, phone call, sent SMS, information leak are suitable feature data for classification malware. The accuracy of our model is 97 % and the FPR is less than 4 %. We also store classification result and confidence value of unknown APK in our database.

Acknowledgments

This research was supported in part by the Ministry of Science and Technology, Taiwan, under the Grants MOST 104-3115-E-007-004 and MOST 103-2221-E-007-073-MY3.

References

[1] "Android SDK", https://developer.android.com/sdk/installing/adding-packages.html.

[2] P.Lantz, "An Android application sandbox for dynamic analysis", Master's thesis, Department of Electrical and Information Technology, Lund University, Lund, Sweden, 2011.

[3] Michael Spreitzenbarth, Felix C. Freling, Florian Echtler, Thomas Schreck, Johannes Hoffmann, "Mobile-sandbox: having a deeper look into Android applications", SAC '13 Proceedings of the 28th Annual ACM Symposium on Applied Computing pp 1808-1815, 2013.

[4] Michael Spreitzenbarth, Thomas Schreck, Florian Echtler, Daniel Arp, Johannes Hoffmann, "Mobile-Sandbox: combining static and dynamic analysis with machine-learning techniques", International Journal of Information Security, Volume 14, Issue 2, pp 141–153, April 2015.

[5] Li-Luen Tsai, "An android machine learning malware detection system using the result of static analysis and dynamic analysis as the features," Master's thesis, National Chiao Tung University, 2014.