Prevention of Malicious Attack on Smart Phones using Amendment Capture Service

S. Fouzul Hidhaya
B.S. Abdur Rahman University
Chennai, Tamil Nadu, India

Angelina Geetha
B.S. Abdur Rahman University
Chennai, Tamil Nadu, India

ABSTRACT
Sensitive data and information is at a greater risk due to the exploitation of the smart phones by Malwares. Malwares that once targeted the computers are now fully designed to extract all the information possible from the mobile phones. This work focuses on preventing the malicious attack on the mobile phones by using Amendment capture service, which checks the smart phone periodically or when a change has taken place and executes the DtKFc algorithm to detect the malicious applications.

General Terms
Machine Learning, Decision tree Algorithm, K-fold Algorithm, Amendment Capture Algorithm.

Keywords
Android system, Application Permission, Android Security, Malicious Applications, Smart phone security

1. INTRODUCTION
Smart phones with its phenomenal growth has brought along security threat as a compliment. Smart phones have become its user's identity. All the personal information like messages, credentials, contacts, photos, videos, etc… are stored in the phones. And this makes security of mobile phone applications gain importance. These applications either have vulnerable codes or they have code to exploit vulnerable Apps. Study shows that the number of new malicious attack samples in 2012 increased 8 times more than those available in 2011.

Smart phone attack can be classified into 4 types [1] as

1. OS attack – vulnerable points in Operating system
2. Mobile App Attack – poor coding of applications
3. Communication network attack – through Wi-Fi and Bluetooth
4. Malware attack – Malicious coding written to exploit system.

One of the most affected platforms is the Android OS. Kaspersky Lab as reported to have found 99% of newly discovered mobile malicious program to target android platform. A survey has found that Android is the most popular platform used by application developers. 71% of the developer population uses Android OS [2].

The android operating system is designed to provide security between two applications. The inter communication between applications is restricted with permissions. These permissions for inter-communication are obtained during the installation process of the application. Normally these permissions are presented to the user in the 'Terms and Conditions' page were the user clicks on 'I Agree' and install the application without reading the terms. Even if the permissions are read, the permission package should be accepted as a whole, one cannot delete certain permissions from the list.

This creates a loophole in the OS. Malicious software makes use of this loophole to steal the data from the user. When the application is first installed the malware cannot be found. The applications can get its malicious code as a patch or an update file from its server. Thus a application may be benign when first downloaded but may turn malicious after an update of the version. So constant monitoring of the system is required to secure data in a android system.

In this work abnormality in the behavior of the smart phone is checked. The Android system is trained and tested using DtKFc algorithm. The program constantly monitors all the applications for an update or patch file. The history of the updated applications is then scrutinized to recognize if the system exhibited an abnormal behavior or a temporal pattern while running the application. The abnormality in the behavior can be recognized using the Decision Tree algorithm and a pattern can be recognized using a Knowledge Based Temporal Abstract Pattern algorithm. If this application is found malicious then a warning message is given to the user.

The rest of the paper is as follows: Section 2 describes the related work in the area, Section 3 describes the system design, Section 4 evaluates and analysis the result and Section 5 concludes the paper.

2. RECENT WORKS
T. K. Buennemeyer et al. [3], in their work introduce capabilities developed for Battery - Sensing Intrusion protection system (B-SIPS) that raises an alert when an abnormal current change occurs. They have developed a Correlation Intrusion Detection Engine (CIDE) that provides power profiling for mobile devices and a correlated view of B-SIPS and Snort alerts. The B-SIPS client is designed with customizable features to accommodate varying user skill levels. Users with advanced computer skills can configure the application to provide more refined detection and alert information, while basic users can effectively operate the system with default settings.

Abijith Bose et al., [4] represented the behavior of malwares based on key observations. Using these observations they have proposed a framework to detect mobile worm, Virus and Trojans instead of signature based solutions. This work also defines various rules to identify the inter-process relationships.

Shabtai et al. [5] designed a machine learning system taking into consideration various features of the system like the messaging, battery usage, CPU usage etc.

E. Menahem et al. [6] have used classification algorithms for machine learning process. Here the behavior of the application during execution is studied, and classifiers are

used to learn patterns in order to classify applications as malicious or benign.

Garcia-Teodoro et al. [7] has given a detailed survey on the techniques used in anomaly- based network intrusion detection. They have classified the different techniques that could be used for intrusion detection.

Griffin et al. [8] have used Signature-based method that depends on the identifying unique signatures that define the malware. This work uses string signatures, each of which is a contiguous byte sequence that potentially can match many variants of a malware family. But this can be applied only to known malicious code.

SCanDroid [9] does a modular analysis to allow incremental checking of applications as they are installed on an Android device. It extracts security specifications from manifests that accompany such applications, and checks whether data flows through those applications are consistent with those specifications.

A. Shabtai et al., [10] in their work does a assessment of the security in the android framework. To identify high risk threats they have conducted a methodological qualitative risk analysis.

A. Shabtai [11] work does a static analysis of Android application files. They have developed a Knowlege based Temporal Abstraction (KBTA). Using KBTA, continuously measured data (e.g., number of running processes) and events (e.g., software installation) are integrated with a temporal abstraction knowledge-base; i.e., a security ontology for abstracting higher-level, meaningful concepts and patterns from raw, time-oriented security data, also known as temporal abstractions

A. Shabtai [12] proposes the implementation of SELinux in Android in order to harden the Android system and to enforce low-level access control on critical Linux processes that run under privileged users. By choosing this route, we can protect better the system from scenarios in which an attacker exploits vulnerability in one of the high privileged processes.

Aubrey-Derrick Schmidt [13] in their work, the features that describe the state of the device are extracted and these features are processed in a remote server. This will be used for anomaly detection.

Jacob et al. [14] gives a survey of different reasoning techniques used by behavioral detectors. They have classified these detectors into main families- stimulation based detectors and formal detectors depending on their data collection and data interpretation mechanisms.

3. SYSTEM ARCHITECTURE

The system works as a service for the android system. This service is provided with highest priority and is given the privilege to interrupt any process in the android system. Figure 1 shows the system architecture.

The system consists of two stages:

1. The preparatory Phase
2. The monitoring phase

3.1 The Preparatory Phase

In this phase the Android system is trained and tested using DtKFc Algorithm and KBTAP Algorithm. There are three main functions in this phase.

1. Feature monitoring
2. DtKFc Algorithm
3. KBTAP Algorithm

3.1.1 Feature monitoring

For learning the behavior of a system some features of the system are monitored. These features are assigned an upper and lower threshold values. If the feature values are between these thresholds then the system is considered to behave normally. If there is a break in the threshold values then there may be a possibility of malicious application running in the system. The features and their parameters that are used to diagnose abnormal behavior are given in table 1.

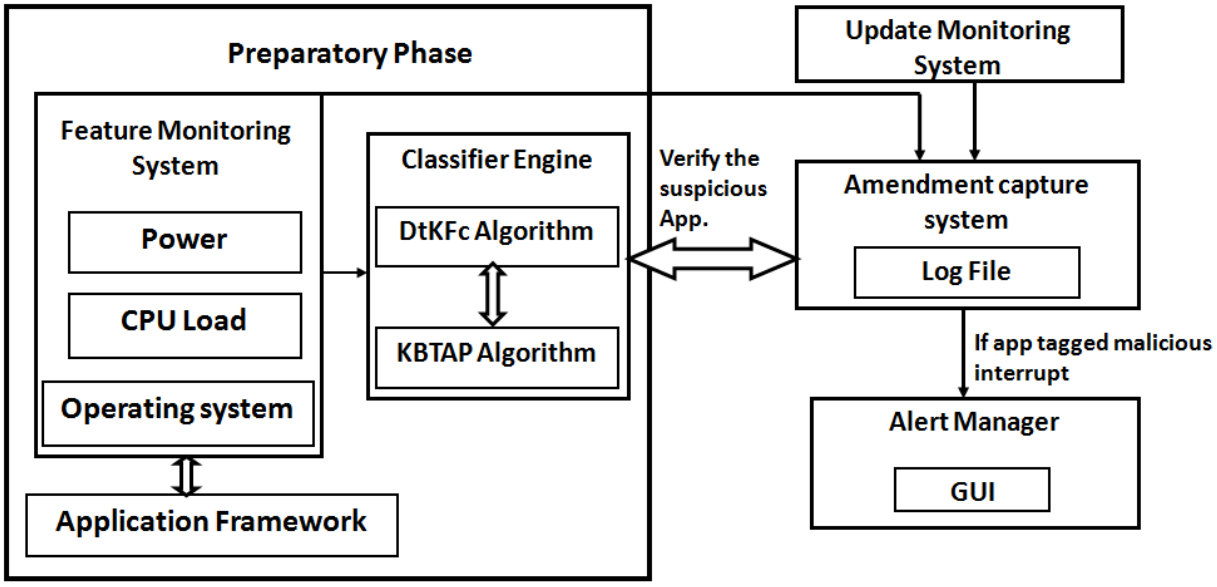

Figure 1: System Architecture

Table 1. Feature Parameters

Feature	Parameters
CPU	CPU usage
	Runnable entities
	Total entities
Battery power	Battery voltage
	Battery Current
	Battery Temperature
Operating System	Process created
	Running process
	Context switch
Messaging Behavior	Similar time frame
	No. of occurrences
	Size and content of Messages

3.1.2 DtKFc Algorithm

Two algorithms, Decision tree and K-fold algorithm are combined to train and test the data sets and to produce efficient results. The decision tree algorithm takes the feature parameters as input and depending on their values classifies them as benign or malicious. The system is trained using decision tree algorithm. Both benign and malicious applications are provided for training. Simultaneously, the system is tested using the K-Fold algorithm. The given dataset of applications are divided into K sets. Each set will have a combination of both benign and malicious applications. The testing is done in K iterations. In each iteration K-1 sets are taken for training. The remaining 1 set is taken for testing the system. The accuracy of the results is improved by this method. This method is named 'the Decision tree and K-fold Combination algorithm' or the DtKFc Algorithm. Figure 2 describes the training and testing process.

3.1.3 KBTAP Algorithm

Apart from the feature parameters abnormal behavior may occur in a pattern. For example, when an application is opened, there may be a sudden increase in CPU process followed by access to the SD card, followed by connecting to Wi-Fi and there may be increase in number of packets sent out. Another Example pattern, Camera application may be initiated and photos may be taken, followed by connecting to the Wi-Fi and the number of packets sent out may increase. All possible patterns are abstracted and are placed in knowledge based temporal abstract pattern [11]. DtKFc and KBTAP algorithms are clubbed together into the classifier engine.

3.2 The Monitoring Phase

3.2.1Amendment Capture Service (ACS)

All the applications downloaded from the Google play store are verified by the Google Bouncer service. The Google Bouncer checks the apps for behavior and malicious code. Only if the app is found benign it uploads the app to the store. But those apps only upload the basic part of the software in the Google play store. The application after getting installed in the mobile phone tends to download the remaining major part of the program from its remote server. So the part of the program downloaded after installation through Wi-Fi or GPRS is viable to have malicious code. The same thing is possible when an application receives a patch code or a update from its server after installation. So an application that is thought to be benign may change malicious after updates.

To protect the mobile system from this situation, constant monitoring of the system is needed. The feature monitoring system keeps a constant watch for abnormal behavior of the system. An update monitoring system watches over the updates and patch codes. These monitoring systems trigger the Amendment Capture Service (ACS) when abnormality arises. ACS maintains a log file which records the history of all the applications in the android system.

The log file has the following details:

1. The installation time stamp of each application.
2. The no. of services triggered by each application.
3. The no. of updates received by each application.
4. The time stamp of the latest version update received by each application and a 'yes' tag to indicate that the update has been checked by the ACS or a 'No' tag if the application is not checked.
5. The time stamp of the latest patch code received by each application.
6. The time stamp of the latest check done by ACS.
7. The no. of times each application was called over the past 2 weeks

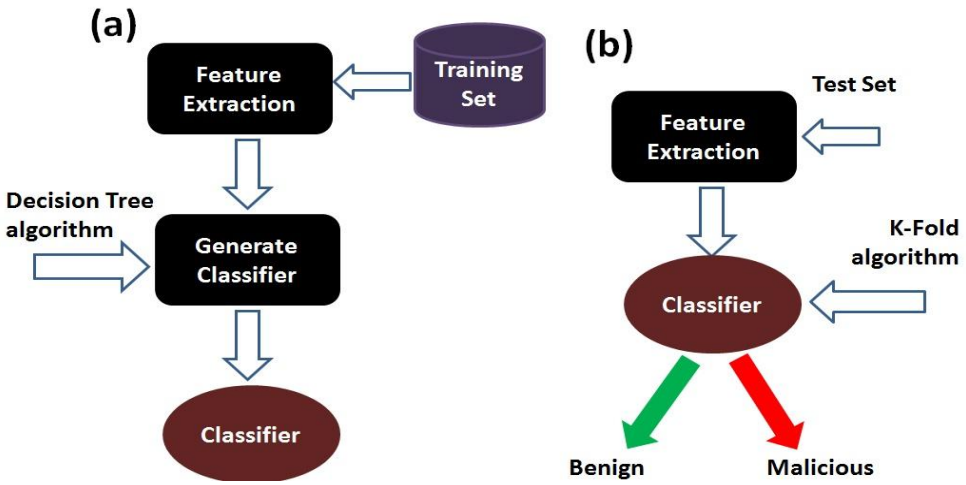

Figure 2: (a) Training using Decision Tree Algorithm (b) Testing Using K-Fold Algorithm

8. The time stamp of the last time each application was opened.
9. The data from feature monitoring system.

ACS is triggered during one of the following situations:

1. When an application is first installed in the mobile phone.
2. When an abnormal behavior has been noted.
3. When an application has received an update.
4. When an application has received a patch code.
5. Periodical check – After every 2 weeks, the time for the check can be chosen by the user.

3.2.2 Amendment Capture Algorithm (ACA)

A change or periodical check triggers ACS. ACS implements the Amendment Capture Algorithm (ACA). Figure 3 describes the Amendment Capture Algorithm.

Step 1:
Capture trigger reason – *trig_reason* ;
Step 2:
If *trig_reason* = *update_monit* then
While (not empty of *no_of_apps*)
Check log file if (*app_update* = true)
Add application to *app_array[ ]*;
app_array[ ] = *list_of_apps_with_update* ;
Return *app_array[ ]*;
Go to Step 3;
Else If *trig_reason* = *Feature_Monit* then
app_array[ ] = list of active apps
Return *app_array[ ]*;
Go to Step 3;
Else If *trig_reason* = *period_check* then
Get (*check_date* = *last_check_date*)
While(*timestamp_app_install*>*check_date* **or**
timestamp_app_update >*check_date* **or**
timestamp_app_patchcode > *check_date*)
Add the application to *App_array[ ]*;
Return *app_array[ ]*;
Go to step 3;
Step 3:
Classifier engine (*app_array[ ]*);
If (app = benign)
Tag the *update_check* = yes;
Else If (app = malicious)
Prompt the user – *remove_application*;

Figure 3: Amendment Capture Algorithm

4. EVALUATION AND RESULTS

The system was stimulated using an Eclipse Emulator. A total of 50 applications were considered as dataset. The dataset has 25 benign applications and 25 malicious applications. The benign applications were downloaded from the android market. To be sure that the benign applications are virus and worm free, they were tested using kaspersky for mobile. The malicious applications were downloaded from contagion.com. The files with .apk and .jar were extracted.

These applications were divided into 5 sets. Each set had a mix of both benign and malicious applications. 4 folds were used for training the system and the remaining 1 set was used for testing. In the next iteration the tested set was included as the training set and 1 set from the prior training set becomes the testing set. Each application was made to run in the emulator for 10 minutes. The accuracy calculator calculated the Accuracy of the result for each test. True Positive Rate (TPR) measure, which is the proportion of positive instances classified correctly. False Positive Rate (FPR), which is the proportion of negative instances misclassified. Total Accuracy measures the proportion of absolutely correctly classified instances, either positive or negative. TP is number of positive instances classified correctly. FP is the number of negative instances misclassified. FN is the number of positive instances misclassified. TN is the number of negative instances classified correctly.

The accuracy calculator uses the formula,

a. TPR = TP /(TP + FN)
b. FPR = FP / (FP + TN)
c. Total Accuracy = (TP + TN)/ (TP + TN + FP + FN)

The training and testing results are as tabulated in table 2. The results from the monitoring system are tabulated in table 3.

5. CONCLUSION AND FUTURE WORK

In this paper, a technique to train the system using DtKFc algorithm and KBTAP algorithm to detect malware is presented. The system captures the changes in the Mobile OS and evaluates the system periodically. The accuracy of our system is calculated using the accuracy calculator and the results shows an average accuracy of 0.94. The system produces most accurate results for the given dataset of applications. At any point of time, the system will be able to find accurately if a malicious application is running in the system. And if a malware is suspected to be present an alert will be given to the user through a Graphical user interface. The assumption taken is that the application runs in the system for at least 5 minutes. This time is needed for the system to recognize the malicious action.

Table 2: Training and Testing Results

Experiment Set	TP	TN	FP	FN	TPR	FPR	Accuracy
I	4	4	1	1	0.8	0.2	0.8
II	7	3	0	0	1.0	0.0	1.0
III	4	6	0	0	1.0	0.0	1.0
IV	3	6	1	0	1.0	0.142	0.9
V	6	4	0	0	1.0	0.0	1.0

Table 3: Trigger Results

Triggers	TP	TN	FP	FN	TPR	FPR	Accuracy
Updates	6	4	0	0	1.0	0.0	1.0
Behaviors	3	4	0	1	0.75	0.0	0.875

6. REFERENCES

[1] http://www.checkmarx.com/2013/11/29/10-challenges-of-mobile-security/?goback=%2Egde_36874_member_5812209353184280577#%21

[2] http://www.infosecurity-magazine.com/view/30153/99-of-mobile-malware-targets-android/

[3] Buennemeyer, T. K., Nelson T.M., Clagett L.M., Dunning J. P., Marchany R.C., Tront J.G., 2008. Mobile device profiling and intrusion detection using smart batteries. In International conference on system sciences , p. 296–296.

[4] Bose,A., Hu, X., Shin, K. G., Park, T., 2008. Behavioral detection of malware on mobile handsets. In Proc. of the 6th international conference on mobile systems, applications, and services

[5] Asaf Shabtai , Uri Kanonov, Yuval Elovici, Chanan Glezer , Yael Weiss, 2012. "Andromaly": a behavioral malware detection framework for android devices, Journal of Intelligence Information Systems, 38,p. 161-190

[6] Menahem, E., Shabtai, A., Rokach, L., & Elovici, Y. 2008. Improving malware detection by applying multi-inducer ensemble. Computational Statistics and Data Analysis, 53(4), 1483–1494.

[7] Garcia-Teodoro, P., Diaz-Verdejo, J., Macia-Fernandez, G., & Vazquez, E., 2009. Anomaly-based network intrusion detection: Techniques, systems and challenges. Computers & Security, 28(1–2), p. 18–28.

[8] Griffin, K., Schneider, S., Hu, X., & Chiueh, T., 2009. Automatic generation of string signatures for malware detection. In Proc. of the 12th international symposium on recent advances in intrusion detection.

[9] Adam, P. F., Chaudhuri, A., & Foster, J. S., 2009. SCanDroid: Automated security certification of android applications. In proc. of IEEE symposium of security and privacy

[10] Shabtai, A., Fledel, Y., Kanonov, U., Elovici, Y., Dolev, S., Glezer, C.,. 2009 Google Android: A state-of-the-art review of security mechanisms. *arXiv preprint arXiv:0912.5101* .

[11] Shabtai, A., Kanonov, U., Elovici, Y.,. 2010. Intrusion Detection on Mobile Devices Using the Knowledge Based Temporal-Abstraction Method. Journal of systems and Software. 83(8),p. 1524 -1537.

[12] A. Shabtai, Y. Fledel, Y. Elovici., 2010. Securing Android-Powered Mobile Devices Using SELinux", IEEE Security and Privacy, 8(3),p.36-44, http://doi.ieeecomputersociety.org/10.1109/MSP.2009.144

[13] Aubrey-Derrick Schmidt , Frank Peters, Florian Lamour, Sahin Albayrak.2009. Monitoring smart phones for anomaly detection. Mobile Networks and Applications 14(1), p. 92-106.

[14] Jacob, G., Debar, H., & Filiol, E., 2008. Behavioral detection of malware: From a survey towards an established taxonomy. Journal in Computer Virology, 4, 251–266.