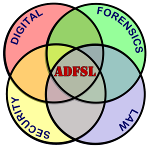

Annual ADFSL Conference on Digital Forensics, Security and Law

2018 Proceedings

May 17th, 3:20 PM - 3:55 PM

Precognition: Automated Digital Forensic Readiness System for Mobile Computing Devices in Enterprises

Jayaprakash Govindaraj
Indraprastha Institute of Information Technology, New Delhi, India, jayaprakashg@iiitd.ac.in

Robin Verma
Indraprastha Institute of Information Technology Delhi, robinv@iiitd.ac.in

Gaurav Gupta
Ministry of Electronics and Information Technology(DeitY), Government of India, gupta.gaurav@deity.gov.in

Follow this and additional works at: https://commons.erau.edu/adfsl

Part of the Computer and Systems Architecture Commons, Forensic Science and Technology Commons, and the Information Security Commons

Scholarly Commons Citation

Govindaraj, Jayaprakash; Verma, Robin; and Gupta, Gaurav, "Precognition: Automated Digital Forensic Readiness System for Mobile Computing Devices in Enterprises" (2018). *Annual ADFSL Conference on Digital Forensics, Security and Law*. 11.
Digital Forensic Readiness, Machine Learning, APK analysis, IPA analysis, Mobile Application security analysis

This Peer Reviewed Paper is brought to you for free and open access by the Conferences at Scholarly Commons. It has been accepted for inclusion in Annual ADFSL Conference on Digital Forensics, Security and Law by an authorized administrator of Scholarly Commons. For more information, please contact commons@erau.edu.

EMBRY-RIDDLE
Aeronautical University
SCHOLARLY COMMONS

(c)ADFSL

PRECOGNITION: AUTOMATED DIGITAL FORENSIC READINESS SYSTEM FOR MOBILE COMPUTING DEVICES IN ENTERPRISES

Jayaprakash Govindaraj[1], Robin Verma[2], and Gaurav Gupta[3]
[1, 2]Indraprastha Institute of Information Technology Delhi, New Delhi, India
[3]Ministry of Communication and Information Technology, New Delhi, India
{jayaprakashg, robinv}@iiitd.ac.in, gupta.gaurav@meity.gov.in

ABSTRACT

Enterprises are facing an unprecedented risk of security incidents due to the influx of emerging technologies, like smartphones and wearables. Most of the current mobile security systems are not maturing in pace with technological advances; they lack the ability to learn and adapt from the past knowledge base. In the case of a security incident, enterprises find themselves under-prepared for the lack of evidence and data. The systems are not designed to be forensic ready. There is a need for automated security analysis and forensically ready solution, which can learn and continuously adapt to new challenges, improve efficiency and productivity of the system. In this research, the authors have designed a security analysis and digital forensic readiness system targeted at smartphones and wearables in an enterprise environment. The proposed system detects applications violating security policies, analyzes Android and iOS applications to identify possible vulnerabilities on the server, and applies machine learning algorithms to improve the efficiency and accuracy of vulnerability prediction. The system continuously learns from past incidents, and proactively collects required information from the devices which can help in digital forensics. Machine learning techniques are applied to the set of features extracted from the decompiled mobile applications and applications classified based on consisting of one or more vulnerabilities. The system was evaluated in a real-world enterprise environment with 14151 mobile applications and vulnerabilities was predicted with an accuracy of 94.2%. The system can also work on virtual instances of the mobile devices.

Keywords: Digital Forensic Readiness, Machine Learning, APK analysis, IPA analysis, Mobile Application security analysis

1. INTRODUCTION

Around 50% of the world population is using the Internet as per the Internet World Stats report 2016 (Miniwatts, 2016). On-line social networks (OSN) take the biggest share of Internet users, where about 2.3 billion users were active on social networks in 2016 (Statista, 2016b). Out of the digital equipment that people use to connect,

© 2018 ADFSL

mobile devices have the largest user base worldwide. The global population uses mobile applications for a variety of activities like social networking, online shopping, mobile banking, and for storing personal as well as official data. The Google play store recorded about 65 billion app downloads whereas over 140 billion downloads from the Apple App store in 2016 (Statista, 2016a). The numbers are expected to further increase as indicated by one of the Gartner's report that has predicted the total connected devices to touch 20.8 billion by 2020 (Gartner, 2015).

The statistics are not different in the enterprises around the world. As per another Gartner's report, by the end of 2017 around 50% of employers would adopt BYOD (Bring Your Own Devices) mainly constituting mobile devices (Gartner, 2013). The increasing adoption of smartphones and wearables within enterprises would bring in additional attack vectors. Hackers and Cyber-terrorists could easily target the vulnerabilities of mobile applications running on such BYOD devices to gain access into the respective organizations. The vulnerabilities could also be exploited for launching attacks, stealing data, bringing down organizational services and to carry out other acts of hacktivism. The BYOD adoption in enterprises would also elevate security risks which arise from the Insider threats. As per Checkpoint's survey report in 2014, around 82% of organizations and individuals observed a rise in mobile security incidents (Checkpoint, 2014), which is expected to increase every year.

There is a need for an intelligence-driven system, which can perform proactive security analysis and be forensic ready. The need has led to the proposal and development of a machine learning based system which the authors have named as 'Precognition.' The proposed system combines both a proactive security analysis as well as forensic readiness into one solution that protects all mobile computing devices within respective enterprises. The authors would like to highlight following contributions of the precognition system:

1. The system can monitor mobile apps installed on a device, or an equivalent virtual instance, in accordance to the enterprise defined security policies.

2. The system can automate the security analysis of the Android as well as iOS mobile application packages to identify potential vulnerabilities.

3. The system uses machine learning for predicting such vulnerabilities to increase the efficiency and accuracy of the solution.

4. The system is digital forensic ready, i.e it collects data required for digital investigations. If an attack happens that exploits a particular vulnerability, the forensic investigators would have quick access to potential pieces of evidence against the attacker.

5. The system was evaluated in a large enterprise in a live environment. The authors have collected and analyzed 14151 mobile apps.

6. The system provides insights based on industry verticals, app categories, and vulnerabilities distribution. These insights could be beneficial for the enterprises and application owners.

The rest of this paper is organized as follows: Section 2 presents the related work in this area, Section 3 describes the Precognition solution system architecture, Section 4

 © 2018 ADFSL

describes the implementation, Section 5 provides the results and inferences, Section 6 provides the conclusion, and Section 7 wraps up the paper with future directions of the current work.

2. RELATED WORK

Digital Forensic Readiness The idea of digital forensic readiness (DFR) was first proposed by Tan, J. The author defines the objectives and measures of DFR when incorporated would increase the forensic readiness (Tan, 2001). Grobler et al., propose DFR as a best practice component for information security (Grobler & Louwrens, 2007). Rowlingson proposes a ten step process for implementing digital forensic readiness by organizations (Rowlingson, 2004). Endicott-Popovsky et al., discuss about how cyber-attacks target networks to disrupt the services. The authors propose Network forensic readiness as a good measure to implement in enterprises (Endicott-Popovsky, Frincke, & Taylor, 2007). Mouton et al., have proposed DFR for the wireless network and implemented it as an additional layer of protection (Mouton & Venter, 2011). Valjarevic et al., proposed and experimented with a DFR for PKI system. They took into account DFR should enhance the security and at the same time not altering the processes of PKI system (Valjarevic & Venter, 2011). Reddy et al., have proposed DFR for large enterprises (Reddy & Venter, 2013). In (Ruan, Carthy, Kechadi, & Baggili, 2013) DFR was analyzed for cloud computing and few prototypes implemented. Dykstra et al. evaluated some of the existing tools like EnCase in the context of Cloud forensic readiness (CFR). The results showed that existing tools were not reliable for CFR (Dykstra & Sherman, 2012) .

Mobile Digital Forensic Readiness Mylonas et al., have focused on ad hoc acquisition of smartphone evidence. The proactive smartphone forensic investigation scheme consists of three parties, Investigator, Independent authority, and the suspect. Here independent authority is the important entity, one who controls the entire process of evidence collection, storage, and transmission (Mylonas, Meletiadis, Tsoumas, Mitrou, & Gritzalis, 2012).

Mobile Application security analysis ComDroid tool analyzes the control flow of procedures and identifies inter-app communication vulnerabilities (Chin, Felt, Greenwood, & Wagner, 2011). Geneiatakis et al., 2015 combines the outcome of static and dynamic analysis & compare with manifest's permission list identifying whether the Android application has overprivileged permissions or not (Geneiatakis, Fovino, Kounelis, & Stirparo, 2015). Li et al., proposes creating call flow graph and thereby observing the data flow from source to sink to detect data leakage (Li, Bartel, Klein, & Le Traon, 2014). SCANDROID, performs data flow, pointer, and control flow analysis to validate if it complies with the security specifications and certifies whether the application is secure (Fuchs, Chaudhuri, & Foster, 2009). Suzanna analyzes various static analysis techniques for Android applications based on Confidentiality, Integrity and Availability (Suzanna Schmeelk, 2014). Enck et al., decompile the Android apps to recover Java source, performs static code analysis using Fortify SCA tool to identify security issues (Enck, Octeau, McDaniel, & Chaudhuri, 2011). Stowaway, a tool to determine API calls and map them to the permissions for finding overprivileged permissions in Android applications (Felt, Chin, Hanna, Song, & Wagner, 2011). Droid test, a server side black box test tool which examines the inputs and output by running set of test cases and correlated test cases

and identifying if there any data leakage (Rumee & Liu, 2015). Droid watch, a prototype enterprise monitoring system for the Android apps for continuously collecting data of interest and perform analytics using tools like Splunk (Grover, 2013). Guido et al., proposes identifying mobile malware on enterprise devices, based on the changes occurring to device partitions, extracting only changed blocks of data and reconstructing images from them (Guido et al., 2013). Shabtai et al., propose applying machine learning techniques to set of features for Android applications to classify whether the application is malware or not (Shabtai, Fledel, & Elovici, 2010).

Following are the gaps we identified in the current literature survey: There is no well-defined enterprise ready practical DFR system for Mobile computing devices, current systems have not leveraged machine learning techniques to improve the accuracy of vulnerability prediction. Most of the current literature is focused on Android and not much work on automating iOS application analysis. Most of the current work on Android require access to source code. In our research work, we have tried to address these gaps and also improve on existing Android security analysis techniques.

3. PRECOGNITION SYSTEM SOLUTION ARCHITECTURE

Precognition system has been designed to work on Android smartphones, Android wear, iOS smartphones, and iWatch. The Precog Client App is installed and run on the devices. The Precog Server processing components and databases are hosted on the enterprise application server or on-premise Cloud. Precog client App monitors the apps installed on the mobile devices and wearables as per security policies defined by an enterprise. The app-package-files and other forensically relevant logs are transferred to the local server, for the apps that violate security policies. In the next step precognition server component running on the server identifies all potential security threats by running both offline-analysis as well as the online-analysis. The offline-analysis is performed by reverse engineering of the application package, whereas the online-analysis is performed by analyzing the app running on the device or an emulator. After the completion of offline-analysis / online-analysis, a list of issues that help in identifying the vulnerabilities is obtained. The authors have trained a machine learning model to predict vulnerabilities based on the list of issues identified in the prior step. The trained model helps in identifying the unforeseen malicious apps running on mobile devices and wearables. The identification of such malicious apps helps in ensuring the forensic readiness of the system. In the case of any security incident, the investigator can use the collected information for conducting a forensic investigation. Figure 1, shows the proposed architecture. The solution architecture consists of six main components:

1. *Mobile computing devices (BYOD environment)* - All the devices are configured to run two instances: a personal and a corporate. The instance could be a virtual image or a sandboxed profile as supported by host operating system. The current solution focuses on the corporate instance on which the Precog Client app is installed. In case of Enterprises that do not enforce the division of profiles, the Precog Client app can be directly installed on the device. The app monitors the device as per the enterprise defined security policies as fol-

 © 2018 ADFSL

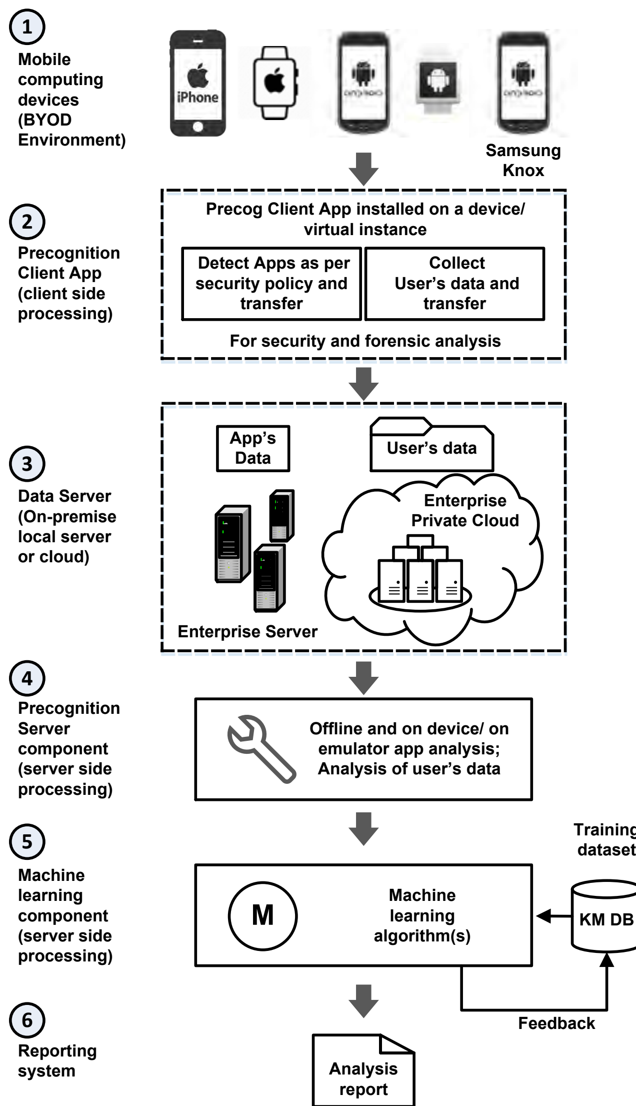

Figure 1. Precognition system solution architecture.

lows:

i All Apps.

ii Apps downloaded from the untrusted marketplaces (Any app not downloaded from the whitelisted app stores. Currently, the solution checks for the Apple App Store, the Google Play Store, the Amazon App store, and the Samsung app store).

iii Apps which need access to special privileges. Currently, the solution checks for apps which require access to SMS, camera, photo gallery, and emails.

iv Virtual instance backups.

v The backup of selected apps user's data.

The first option of monitoring all apps may not be practical for large enterprises, considering the amount of processing required for analysis. However, this security policy can be enforced for a selected group of high-risk individuals (like CEO, COO, and other Senior Management personnel) within the enterprises. The second option of monitoring apps downloaded from untrusted sources is a more practical approach. However, there is a risk of overlooking malicious apps from whitelisted sources. In the real world scenario, the enterprises can setup security policies by adding new, modifying or deleting the existing policies. After the setup, they can configure a combination of various security policies based on the requirements. In the current prototype, the authors have implemented the second option checking for apps from untrusted sources.

2. *Precognition client App (Client side processing)* - All devices have the Precog client app preinstalled, which filters apps according to the enterprise's security policy, collects app-specific data, collect user's data and transferred to the server for further processing.

3. *Data Server* - Stores app and user data for further processing. The app data consists of app-package files (.apk and

.ipa), whereas, the user data includes data generated by the user like call records, SMS, camera photos/videos, audio files, browsing history and contacts.

4. *Precognition Server Component (Server side processing)* - Includes Offline and On Device/Emulator Security Analysis of untrusted apps to identify security vulnerabilities. The selected user's data is utilized for security and forensic investigation.

5. *Machine Learning Component (Server side processing)* Includes a trained learning model, a knowledge database and continuous feedback to the model. The Machine learning algorithms aid to predict vulnerabilities and improve the accuracy and efficiency of the system.

6. *Reporting system* - produces security and forensic analysis reports, that recognize the threat before it can happen, so that the security incident can be prevented.

3.1 Precognition System work-flow

The following section explains the Precognition System Client server workflow.

Client side workflow - the Precognition client app detects apps as per security policy and transfer them to the Server for further analysis. The client app takes the backup of the user app data in case a virtual instance needs to be checked. This backup is stored in the enterprise cloud.

Server side workflow - the Server component runs on the enterprise server, where it is responsible for:

- *App Analysis* - it picks up the uploaded apps and performs the security analysis. Afterwards, it uses machine learning to identify all possible security vulnerabilities and threats.

- *Virtual instance analysis* - In case the check needs to be done for a virtual instance running on a device (like the work-instance on Samsung Knox), the server analyses the app data and the corresponding user's data of the reported app. The analysis happens as per the security policy and follows the same process as done for a standalone device.

- *User's data analysis* - As part of the previous point, a forensic analysis of user data (SMS, Voice, Mail history, Call history) is carried out in the cloud.

- *Incident handling* - If an attack happens to exploit a particular vulnerability, the security analysis information will help in understanding how the attack was carried out. With the collected forensic information, the forensic investigators would be able to link the attack with the evidence against the attacker responsible for it.

4. PRECOGNITION PROTOTYPE IMPLEMENTATION

In this research work, the authors have implemented a working prototype of the proposed solution on Android and iOS devices (phones and wearables). The solution works on the security policy rule of scrutinizing all 'Apps downloaded from the untrusted marketplaces.' The table 1 enlists the devices that the authors have used in the implementation process.

 © 2018 ADFSL

Table 1. List of Devices utilized

Oerating System	Device
Android	Samsung Galaxy S3
Android	Samsung Galaxy S7 with Knox
Android Wear	Asus ZenWatch
iOS	iPhone 4, iPhone 5s
watchOS	Apple Watch Series 1

4.1 Precognition Client App

The Precog Client app for *Android* and *Android Wear* was created in Java; whereas, the authors have used Objective-C to write the app for *iOS* and *watchOS*. The Precog Client app is installed either on the device or the virtual instance. The authors also created a virtual instance test case using **Samsung Knox**, which is a feature available only on specific Samsung devices. When activated, Samsung Knox allows users to run two virtual instances on the phone; one for personal work and the other with a workspace environment dedicated to office work.

4.1.1 Detecting the untrusted .apk/.ipa

Android and Android Watch apps (.apk) - the Precog Client app gets the list of all installed and newly downloaded apps on the device through the `GetInstalledPackages[ ]` command. The authors then validate if the packages belong to any of the following trusted sources based on the app-package name; namely, `com.android.vending` (for the Google Play), `com.amazon.venezia` (for the Amazon Appstore), and `com.sec.android.app.samsungapps` (for the Samsung Galaxy Apps - store).

If the PackageName does not belong to the Google Play store, the Amazon Appstore or the Samsung Galaxy Apps store, then the package is marked as suspected.

iOS and watchOS apps (.ipa) the Precog client app gets the list of all installed apps using the `GetInstalledappbundles[]` command, and scans through the bundles to check if the file `embedded.mobileprovision` exists in the app resource path or not. If the file is present, then the authors consider that app as untrusted and mark it suspected.

In the current prototype implementation, the authors have executed all their tests on a jailbroken iPhone; because of the iOS's sandboxing feature that does not allow one app to access another app's information on any of its devices unless they are jailbroken.

4.1.2 Transferring the suspected apps to external storage system

The Precog Client App process through all suspected apps identified in the previous step 4.1.1. For the *Android devices*, it copies the suspected apps to a folder created on the memory card/ the external storage. Whereas, for the *iOS devices*, it copies all the suspected apps to a predefined private folder on the device, which is easily accessible on a jailbroken device. The copied apps are then compressed into a zip file.

4.1.3 Uploading the suspected/untrusted apps

All the suspected/untrusted apps are available in their respective folders after the previous step 4.1.2 concludes. The authors installed an Apache Server, which is managed by XAMPP app, on the Precog Server where all the suspected/untrusted apps from the *Android devices* are uploaded.

In case of the *iOS devices*, all suspected/untrusted .ipas are uploaded to the Precog Server which the authors installed using an Apache Server managed by XAMPP app on a separate Mac machine. The PHP web service is launched on the server to receive all these uploads.

4.1.4 Uploading user's data to the server

Android virtual instance - the Precog Client app uses tools like *Ultimate Backup Lite, Syncios, and Wondershare MobileGo* to take the backup of user's data (refer to the table 2 for user data categories). The un-rooted device backup contains `app/data/<package name>` which consist of only apk files, whereas, the rooted device backup contains `app/data/data/<package name>`.

A typical Android app contains the following data - Database/ : contains app's database, Lib/: holds libraries and help files, Files/: other related files, Shared_prefs/: all preferences and settings, and Cache/: keeps all caches. In case the device is rooted, SQLite files can be found under `app/data/data/<package name>/database/filename.db`, and the cache files can be found under `app/data/data/<package name>/cache/binary format`. The images can be found for both rooted as well as the un-rooted device under `photo/storage/sdcard0/`.

The data is be stored in XML format; for example, the Contacts/callhistory.xml looks like

```
<contact>
<displayname/>
<givenname/>
.....
<number>8105946466<number/>
<address>Address<address/>
<photo/>data will be in byte
text<photo />
<email/>aaa@gmail.com<email/>
```

Table 2. User data backup

User Data	Files
Video	Storage/sdcard0/ all the video file
Audio	Storage/sdcard/ all the audio file
Photo	Storage/sdcard/all the photos
DCIMPhoto	Storage/sdcard/All images taken by device camera
App	data/app/all apk files
Call History	callhistory.xml Call details and duration of the call
SMS	sms.xml Sms in plain text
Contacts	contacts.xml contact numbers, names & address

4.2 Precognition Server component

The server-side application provides the following two functionalities: carrying out the security analysis of app binary packages, using machine learning for identifying vulnerabilities; and arranging the security incident handling.

Android and Android Wear - the server-side application is created using Java, with the following pre-requisite requirements: Java Runtime Environment (1.8 or above); Environment Variable "JAVA HOME" set to the location of Java executables; Android SDK installed; Environment Variable "ANDROIDHOME" set to the location of Android Platform-tools; Windows Operating System (7 or above); R installed; Rooted Android devices with USB Debugging enabled in Developer Options; and Device driver from manufacturer for the concerned device installed on the system.

 © 2018 ADFSL

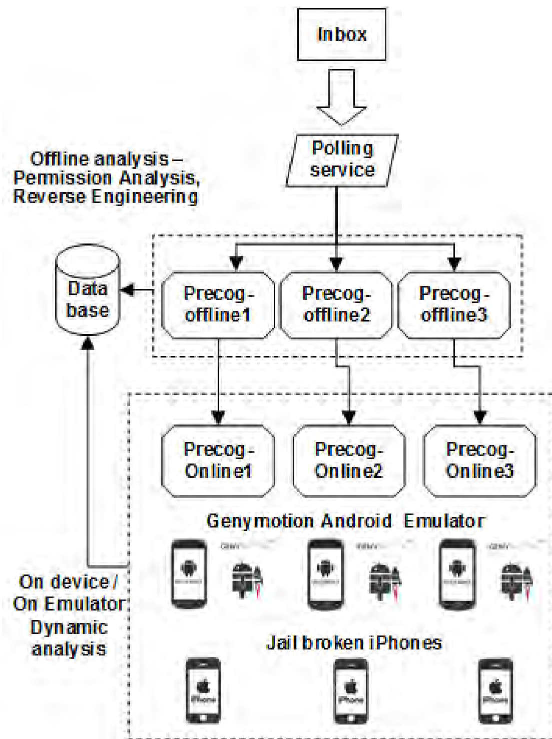

Figure 2. Server Component

iOS and watchOS the server-side application is implemented using Objective-C/Java, with the following pre-requisite requirements: Java Runtime Environment (1.8 or above); Xcode and its command line tools installed; Eclipse IDE installed; a MAC System; and a jailbroken iOS devices with SSH, and SFTP protocols installed on it.

4.3 Server component - Offline and On-device/Emulator security analysis

The server component has following four main modules: 1. Inbox; 2. Polling service; 3. Precog offline module; and 4. Precog online module (refer to figure 2.)

Inbox - receives the untrusted .apk/ipa from the Client-slide component.

4.3.1 Polling Service Module

Android and Android Wear - This module is created as the windows service, which aims to poll the Inbox folder for the incoming untrusted .apk packages and then to hand them over to the Precog Parallel processing component. The Precog Parallel processing component is created using the *ThreadPool* class in C-Sharp to manage multiple threads. The numbers of threads are controlled by two parameters `Threadpool.setMaxThreads` and `ThreadPool.setMinthreads`. In the current work, three apps can be processed simultaneously, and the 4th app will be in the queue until any of the threads become available. However, the maximum number of threads can be set based on available processing power. The untrusted app is then handed over to Precog Offline and Online processing components separately.

iOS and watchOS - the module, which has been created as an Objective-C console application, polls the Inbox folder for incoming .zip files, and subsequently invokes a Java component (jar file) with the current .zip files' paths for offline and online security analysis. The polling service is implemented with *NSOperation* and *NSOperationQueue* of Objective-C for parallel processing and queuing. The polling service will invoke maximum three Java components, i.e., analysis of 3 files can be run parallel and the next zip files uploaded are maintained in a queue

4.3.2 Precog offline processing component (.apk)

Precog offline processing component is a Java application packaged as Precog.jar file which takes the following parameters as inputs and performs .apk permission analysis and Reverse-engineering.

- .apk file's location.
- ClassKeywords.xml - contains a list of sensitive API names and the corresponding threat(s) due to usage of those APIs.
- PPIInfo.xml - contains a list of PII keywords. i.e., password, email, API, license.

Android permission analysis The authors have analyzed 3,622 top rated apps, and created a database of permissions versus apps category for the same. For the selected app, the authors executed batch `adb` commands to get respective apk-information. The list of obtained permissions was compared against the database to identify all possible overprivileged permissions.

Reverse Engineering The untrusted app is reverse engineered to its respective .smali files, java classes, and manifest files. The authors then run a batch execute the APKTool command to decompile the given .apk file to its corresponding .xml file and .smali file.

Afterwards, the authors decomiled the .apk to its java classes using *Dex2jar* and *jad* decompilers. All the source files are then parsed against the previously discussed classkeywords.xml and PPIInfo.xml files. The identified security analysis information is stored in the database. The output is shown in the figure 3; includes the file-path, line number, sensitive information, function name, possible threat based on usage of that function, and the description of the threat.

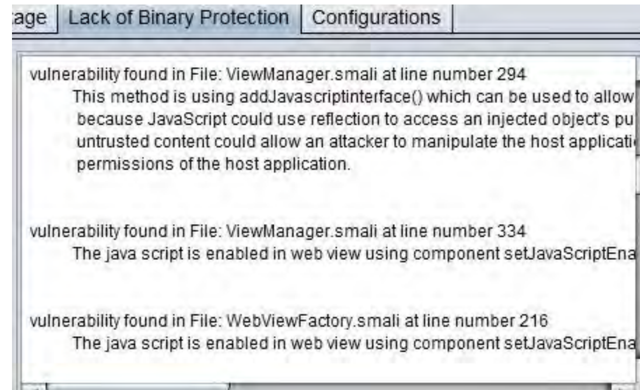

Figure 3. Lack of binary protection.

4.3.3 Precog offline processing component .ipa

Precog offline analysis consists of a Java component that invokes the Java services to perform the reverse engineering of the untrusted app.

Unlike the .apk files which can be decompiled to the get the corresponding source code in one go, the .ipa files need **a two-step process**. The .ipa package is first fed to a set of tools to obtain the corresponding **header files**; which are checked for flaws that result from insecure programming practices, like hard-coded values and use of inherently insecure APIs. The second step involves treatment of the .ipa package through a different set of tools that produce the corresponding **assembly code**; which should be checked for human programming errors and malicious code bytes. Both the steps as mentioned above are required to get a complete analysis of given untrusted .ipa package; hence the authors have performed both whose details are presented in the following paragraphs.

Reverse Engineering .ipa to header files The selected zipped app bundle is unzipped using a shell script to get its resource files and corresponding app binary. Then the app binary is reverse engineered by decompiling the app binary file to its corresponding header files (by using the **ClassDump** command). These header files are then scanned for sensitive keywords, like PII information, hard-coded values, and insecure APIs. The PList and resource files, from the app bundle, are also scanned for such sensitive keywords.

The program then parses through all these

 © 2018 ADFSL

files to find out all potential security flaws related to the sensitive info, PII, or the insecure APIs. The information (file-path, sensitive information) is then stored in the database.

Reverse engineering .ipa to assembly The selected app is reverse engineered by calling a shell script to execute **oTool** command to decompile the binary file to its corresponding assembly code. The parsing is performed throughout the complete assembly code to find out any security flaws related to sensitive info or PII, or any insecure APIs. The respective flaws are then flagged and displayed to the user. The identified information (file-path and sensitive information) is stored in the database.

4.3.4 Precog online processing component .apk

The untrusted .apk package is installed on **Genymotion** Android instance using **adb** commands to perform the dynamic analysis. In the current prototype, three Genymotion instances were set up, to run three threads in parallel. The program takes two inputs - firstly the decompiled .apk folder path, and secondly the XML file containing the PII information to perform the following three types of analysis:

Logcat analysis , which captures the logcat information and parses it for sensitive information disclosure, which is later stored in the database.

Run exported activities - the program fetches the ExportedActivityNames from the manifest file. For each activity in `ExportedActivityNames[]`, a batch execution of `adb shell am -n` starts returning packageName and activity (takes a snapshot of the activity).

Insecure data storage analysis traverses through the application folder to search for any sensitive information (or PII) in the text or the cache or the database files. The output of this step is shown in figure 4. Example:- `/file1/xyz.apk - 123 - Lastname`.

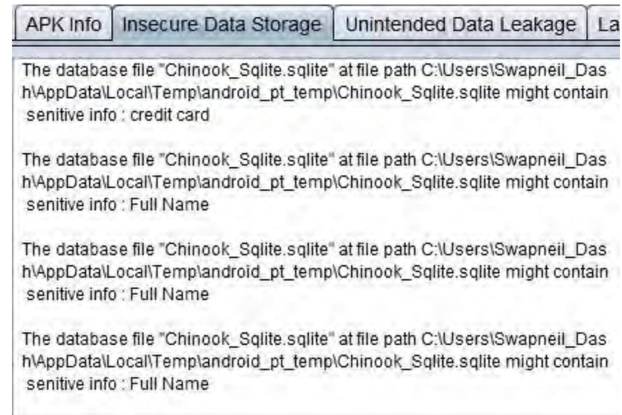

Figure 4. Insecure Data storage.

4.3.5 Precog online processing component .ipa

Since the authors designed the polling service to run three packages in parallel at a time, they used three jailbroken iPhones for performing the online analysis. The jailbroken iPhones were connected to a Mac machine, which was on the same network. The untrusted app is installed and launched on the jailbroken iPhones using SSH and SFTP protocols, and the Xcode commands from the Mac machine. The authors performed the following two types of analysis on the same:

Insecure data storage analysis - The files like the plist and the SQLite databases from the app data folder are transferred to the Mac machine using SFTP commands. Afterwards, the Java component of online analysis is run and all these files are scanned for sensitive keywords, and PII information. The output is shown in the figure 5 is stored in the database.

IP Address '192.168.43.100' found in the application as a Hardcoded String.

IP Address '127.0.0.1' found in the application as a Hardcoded String.

Figure 5. iOS Insecure Data storage

Unintended Data leakage analysis the files like cookies, caches, and the snapshots which are present in the app data folder are transferred to the Mac machine using SFTP commands. Afterwards, the Java component of online analysis is run, and all the files are scanned for sensitive keywords, PII information. The snapshots are copied and displayed to the user for manual analysis. The output is stored in the database.

4.4 Machine learning implementation

The machine learning model created to predict the vulnerabilities in the applications is based on authors previous findings and results of the analysis of such apps. The offline and the online app analysis reports are processed to build the training data, which is used to generate a learning model. Once built, the model can be used to predict the vulnerabilities for a new app. The training data is regularly updated with the predicted values to improve the accuracy of future predictions. The approach consists of the following steps:

Training data - the past security analysis reports generated by the Precog solution are taken as the input for the creation of the training dataset. A set of keywords is used as the features for the training data. These features include any sensitive APIs and keywords which are likely to be present in an Android application. The training data consist of one record per app that were analyzed using the Precog solution.

Learning Model Generation - The authors have used the Naive Bayes algorithm for classifying the apps based on whether the vulnerability is present or absent. The algorithm calculates the probability for each of the categories of the vulnerabilities, and then assigns the vulnerability with the highest probability. One application can have more than one vulnerability, hence the authors choose *one vs. the rest* method for multi-label classification (Lars Shmidt Thieme, 2007).

In multi-label classification, a model is generated for each of the vulnerabilities. For instance, if we have a vulnerability 'Application Misconfiguration,' we use a binary classification approach in which we have only two classes: 0 and 1 where 0 denotes 'Vulnerability Absent' and 1 denotes 'Vulnerability Present.' Similarly, a model is generated for each of the vulnerabilities. Once this process is complete, then the training model can be used to predict the vulnerabilities.

Feedback system - After predicting the vulnerabilities, the record, that contains the features and the predicted values, is updated into the training data. The updated dataset can be used for future predictions.

Predicting vulnerabilities - The test data is input to each of the models for predicting all the vulnerabilities.

The machine learning implementation consists of three program modules. The first one is a Java program which is used for creating the Training data set. The second Java program is used to create Test data for the new app; and The third is a R program that is used for creating the learning model (refer to the machine learning flowchart shown in figure 6).

 © 2018 ADFSL

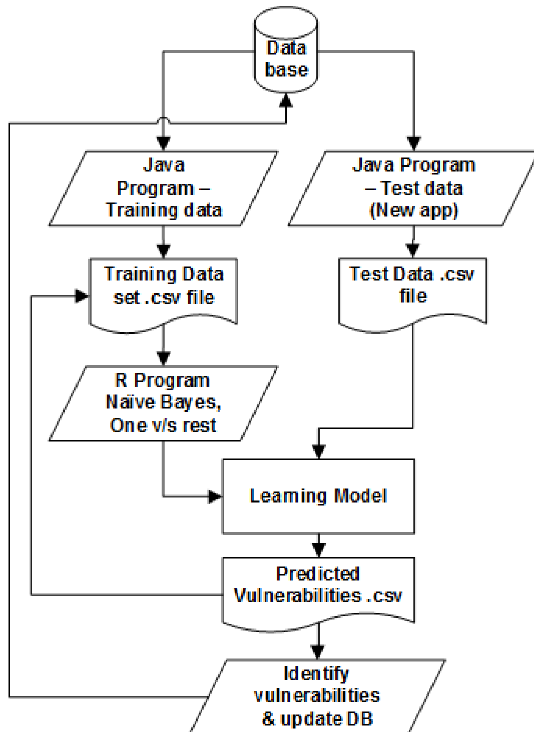

Figure 6. Machine learning flowchart

4.4.1 Training dataset creation (Java method to create training data)

Inputs: all the past analysis security reports from the database and a configuration file with all the keywords to be searched in the report.

Keywords: ADB Backup, TelephonyManager.getdeviceID, setJavaScriptEnabled, getInstallerPackageName, logging some information or data within the code, PRAGMA key, Runtime.getRuntime. exec(...), getInstallerPackageName, addJavascriptinterface(), Keystore, Email, password, credit card, Cookies, Country, android:exported =true, Grades, Zip Code, Full Name, screen name, Gender, Telephone number, Date of birth, Driver's license number, Birthplace, IP address, Login name, fingerprints, Vehicle Registration plate number, Digital identity, Genetic information, Passport, Environment.getExternalStorageDirectory, sendDataMessage, sendMultipartTextMessage, sendTextMessage, setAllowFileAccess, Sqlite, getInstallerPackageName, System.loadLibray, Runtime.getRuntime.exec(...), dexClassLoader(), debug mode is explicitly on, debug mode is explicitly off, sqlcipher, Settings.Secure.ANDROID_ID, setAllowFileAccess(false), MODE_WORLD_READABLE, sharedUserId, Secure. ANDROID_ID, google map.

Method: the Java program reads all the past analysis security reports generated by the Precog solution and generates the training data by parsing for each of the keywords which are stored in the configuration file. It would be run only once for creating an initial training data. If the keyword is present, '1' will be updated, else a '0' is updated. This process repeats for each application security report. For each of the apps, the vulnerabilities are assigned manually by analyzing the app for the present vulnerabilities. The OWASP Mobile top 10 vulnerabilities are considered for classification (refer to the table 4).

The output is a TraningData.csv file that consists of a vulnerabilities list along with the set of keywords for each app.

4.4.2 Test Dataset creation (Java method to create test data)

Inputs: A configuration file with all the keywords to be searched in the report file and the string values from the database.

Method: the method reads the new app security analysis data from the database, then checks for each keyword (P1 to Pn) from the configuration file. If the keyword is present, a '1' otherwise a '0' is updated. The output is a TestData.csv file with a list of all

© 2018 ADFSL

Table 3. Training data set

	Vulnerabilities				Keywords			
AppName	M1	M2	...	M10	P1	P2	...	Pn
aLogCat	1	1	0	1	1	1	0	0
Amazon Shopping	0	1	1	1	0	0	0	0
Angry Birds	1	0	0	0	0	0	0	0
Asphalt	1	1	0	1	1	1	0	0
Book-MyShow	1	0	0	0	1	0	0	0
BOX8	0	0	1	0	1	1	0	0
BusyBox	0	0	0	0	0	0	0	0
Callapp caller ID	0	1	1	1	1	1	0	0
Calorie Counter	1	1	1	1	1	1	0	0
Cam Scanner	0	1	1	0	1	1	0	0
Canara Enfobook	0	0	0	0	0	0	0	0
Candy Crush	0	1	0	1	1	1	0	0
CirclePay	0	0	0	0	0	0	0	0
Clash of Clans	0	0	0	0	0	0	0	0
Cloud-Check-Service	1	1	0	1	0	0	0	0
DB Sum-mit	0	1	1	0	1	1	0	0
Dominos	1	0	1	1	1	1	0	0
Download Manager	0	1	0	1	0	1	0	0
English Grammar	0	1	0	0	1	1	0	0
eRail	1	0	0	0	1	1	0	0
ESExplorer	0	0	1	1	1	1	0	0
Facebook	0	1	0	0	1	1	0	0
Flipkart	1	1	1	1	0	1	0	0
FolderSync	1	0	1	1	1	1	0	0
Food Ordering app	1	1	1	0	1	1	1	0
Gmail	0	0	1	0	1	1	0	0
...	0	0	1	1	0	1	0	0
Zip-Recruiter	1	1	0	1	1	1	0	0

Table 4. OWASP Mobile top 10 vulnerabilities

Code	**Vulnerability**
M1	Weak Server Side Controls
M2	Insecure Data Storage
M3	Insufficient Transport Layer Protection
M4	Unintended Data Leakage
M5	Poor Authorization and Authentication
M6	Broken Cryptography
M7	Client Side Injection
M8	Security Decisions Via Untrusted Inputs
M9	Improper Session Handling
M10	Lack of Binary Protections

Table 5. Test data set

Vulnerabilities					Keywords				
M1	M2	...	M9	M10	P1	P2	P3	...	Pn
0	0	...	0	0	0	1	1	...	1

the keywords for each app. It also has the list of vulnerabilities (M1 to M10) initially marked as 0 (refer to the table **??** for sample Test data set, and refer to the table 4 for vulnerabilities list).

4.4.3 Java-R integration (Java method for calling the R program from Java)

The method first calls, then executes the R program from the Java program for the learning model generation.

The R program The program takes the training data and test data as the input, then applies the Naive Bayes Algorithm to generate a model. The resulting model from the algorithm is used on the test data. The predicted values are stored in a file which is read by the Java program.

The R program then updates the training data using the test data and the predicted values. The Naive Bayes algorithm uses the Bayes theorem for finding the probabilities:

$$p(Ck|x) = \frac{p(Ck)P(x|Ck)}{P(x)} \quad (1)$$

 © 2018 ADFSL

Where, p(Ck) is the probability of a vulnerability being present in the complete dataset. P(x|Ck) is the probability of a particular instance given its vulnerability. P(x) is the probability of an instance given complete dataset. P(Ck|x)is the probability that a particular vulnerability for a given app is present.

We find P(Ck|x) for each of the vulnerabilities. The predicted vulnerability will be the one which has the highest probability. A simple implementation of Naive Bayes can do only multiclass classification; however, an app can have more than one vulnerability, so the authors employ *one vs. the rest* method for multi-category classification, where a model is generated for each of the vulnerabilities. For example: If we have a vulnerability 'Application Misconfiguration,' we use a binary classification approach in which we have only two classes: 0 and 1 where 0 denotes 'Vulnerability Absent' and 1 denotes 'Vulnerability Present.' Following a similar approach, the model is generated for each of the vulnerabilities. The output of this would have the predicted values for each of the vulnerabilities in a comma-separated file. If a vulnerability has a 0 value, then it is not present; otherwise, the vulnerabilities with a value 1 denote their presence.

4.4.4 Predicting Vulnerabilities (Java method to read predicted data)

The method reads the predicted values from the file and maps it to the vulnerabilities, then displays the list of the predicted vulnerabilities.

Input: The file containing predicted values. *Method*: The method reads the data from prediction file which has 0s and 1s for respective absence and presence of each vulnerability. If it is a 1, then the corresponding vulnerability is displayed. *Output*: List of predicted vulnerabilities.

4.4.5 Feedback system (R program)

The R program which is used for predicting the vulnerabilities, after updating the vulnerabilities to the CSV file also updates the training data, forming a feedback system. Hence, the predictor values from the test data as well as the predicted values are updated to the training dataset. Thus, the training data is updated after the prediction of each app. This makes the system to be more accurate by continuously improving the learning model.

5. RESULTS

Machine learning results: In order to measure the predictive ability of the proposed model, the authors have tested the accuracy on a set of data which is not used in the estimation process. The authors call the data as the "Test Set" and that used in the estimation as the "Training Set." The authors have used the Mean Squared Error (MSE) to measure the proposed model's predictive accuracy. The equation is as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2 \qquad (2)$$

where y_i - vector of i predictions, $\tilde{y}_i$ - vector of observed values.

In the above situation, "leave one out cross-validation" is one of the approaches that can be utilized. There are two modifications of this technique, namely **leave-k-out cross-validation**; where k observations are left out at each stage, and **k-fold cross-validation**; where the original data is randomly divided into k sub-datasets, and one is left out in each iteration. In the current implementation, the authors have realized a **k-fold cross-validation** for testing the accuracy of the proposed model.

To obtain the accuracy measures, the authors have considered n independent observations, y1, y2, y3, , yn; where k = 5. The process is explained in the following steps:

1. Partition the dataset into k equal parts, say (k1, k2, k3, k4, k5).

2. Let observation set ki form the test set, and fit the model using the remaining data. Then compute the error ei* for the omitted observations. The result is called predicted residual.

3. Repeat step 1 and 2 for i=1, 2, 3, ...,k.

4. Compute the MSE from e1*, e2*, e3*, ...,ek*. The authors call it the CV.

The authors divided the training data into two sets, namely the training data set and the test data set. The training data set forms 80% of the total volume of the data, while the test data is the rest 20%. The variable “k” specifies the number of times the cross-validation approach is used. In each round, the authors took a different set of test data (20% of the total volume) such that no instance of data gets repeated for the following rounds. The no-repetition policy ensures that the algorithm’s accuracy gets tested on different kind of data in each iteration. The authors got a mean accuracy of 94.2%, which was obtained by considering the prediction-accuracy of all the vulnerabilities followed by taking the average of the same. The authors have also included a feedback system that updates the training data which in turn increases the accuracy of the algorithm’s prediction. The table (6) presents the accuracy figures obtained while predicting the OWASP Top 10 vulnerabilities.

The authors have tested the system on 14,151 apps, which includes both Android (.apk) as well as iOS (.ipa) apps. Top Apps belonging to various industry vertical and application categories were downloaded for the test. The figure 7 shows the vulnerabilities distribution, where the word Domains refers to Industry verticals under which a particular app falls, and the word Category denotes the group name assigned to the particular app by its developers/owners. The authors observed that Shopping, Travel & Local, Business, Finance, Lifestyle are the top five categories which are most vulnerable to exploits. The figure 8 shows the distribution that depicts domain versus the types of vulnerabilities. The authors observed that the **Lack of binary protection**, **Insufficient transport layer protection**, and **Unintended data leakage** are the top three vulnerabilities across all the domains (refer to figure 10 for details. Moreover, **Retail**, **Media & Entertainment**, **Eduction**, and **Finance** are the top four sectors which are vulnerable to exploits (*these sectors have been easy prey for the hackers*). The table 4 provides the Naming convention for the Application Vulnerabilities.

Table 6. Accuracy of vulnerabilities predicted

Vulnerability	**Accuracy**
Weak Server Side Controls	100%
Insecure Data Storage	93%
Insufficient Transport Layer Protection	86%
Unintended Data Leakage	93%
Poor Authorization and Authentication	85%
Broken Cryptography	100%
Client Side Injection	92%
Security Decisions Via Untrusted Inputs	100%
Improper Session Handling	100%
Lack of Binary Protections	93%
Average	94.2%

 © 2018 ADFSL

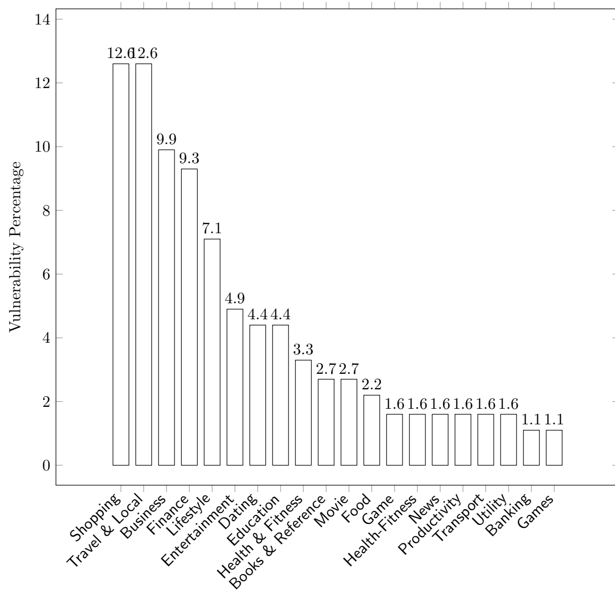

Figure 7. Vulnerabilities distribution by categories

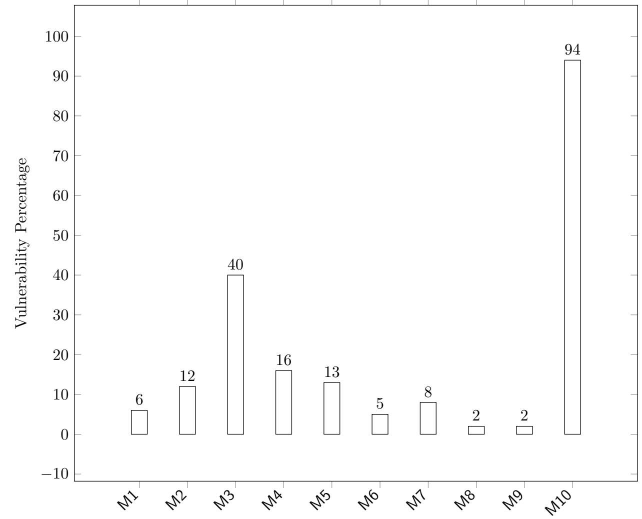

Figure 9. Vulnerabilities distribution

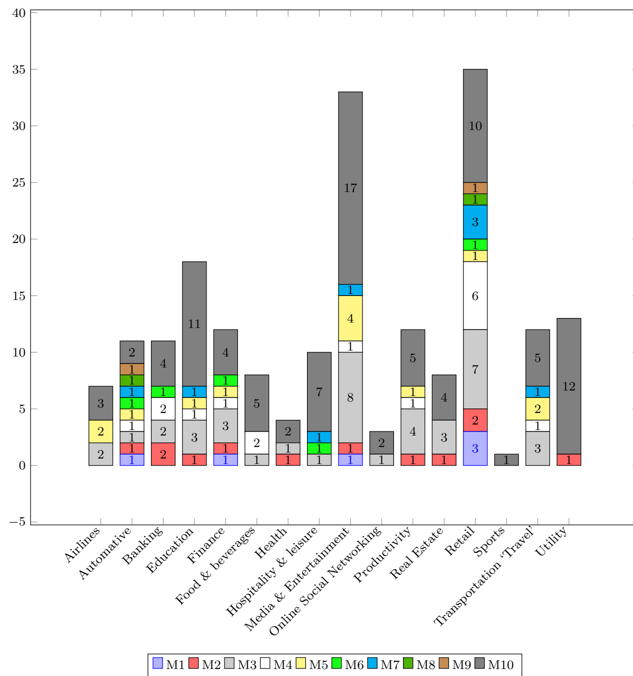

Figure 8. Domains versus types of vulnerabilities

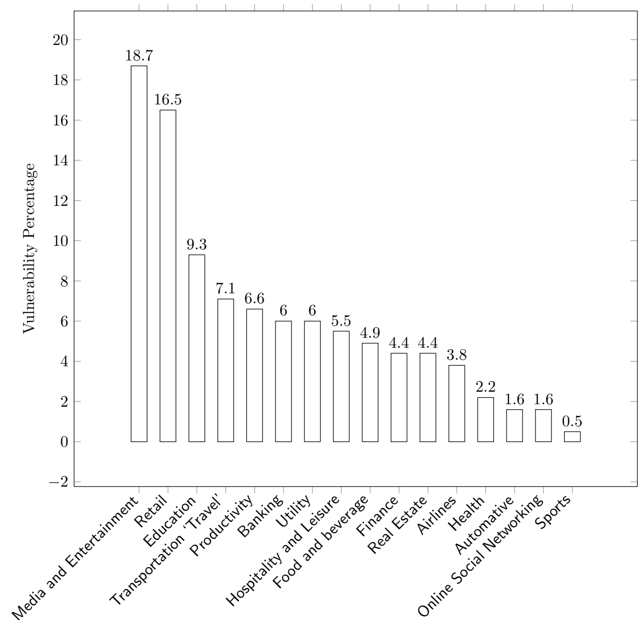

Figure 10. Vulnerabilities distribution by Domains

6. CONCLUSIONS

The authors have successfully created and demonstrated a practical *'Security Analysis and DFR system for Mobile computing devices in an Enterprise environment.'* The proposed system is capable of identifying an app which is downloaded on an Android (phone, VM, or wearable) or an iOS (iPhone or iWatch) from an untrusted source. After identification, the system transfers the tainted app to a pre-deployed server for its security analysis which aims to identify top security threats that could have been exploited by hackers or malicious entities. The solution then offers a process to automate the security analysis, and suggested potential improvements to the existing app forensic analysis techniques. The authors have obtained an prediction accuracy of 94.2 %, after applying machine learning techniques to predict vulnerabilities. The authors found that **'M10-Lack of Binary Protection'** and **'M3-Insufficient Transport Layer Protection'** are the most frequently encountered security threats because the application developers do not perform certificate inspection and fail to incorporate preventive measures against reverse engineering of their apps. Moreover, the apps belonging to **'Retail'** and **'Finance'** sectors were found to be highly vulnerable primarily because these apps have high exposure to customer's PII information in addition to finance related information like bank account and card details. The situation gets critical with the increasing volume of customers who have started using these apps from the above-stated domains, in turn attracting more number attackers.

7. FUTURE WORK

The authors have identified following areas for further research:

1. Implement the DFR system for other security policies apart from apps downloaded from the untrusted marketplaces.
2. Identify tainted virtual instances of the corporate profile, initiate the efficient and incremental backup of the image, perform security and forensic analysis of the image on the cloud server.
3. Automated detection and handling of security incidents and respond dynamically.
4. Machine learning techniques for identifying malicious activities, device / user behavior analysis and security incident handling.
5. Implement Anti Anti-forensics techniques to counter any Anti-forensic attacks.
6. Integration with a privacy preserving framework system.

REFERENCES

Checkpoint. (2014, October). *The impact of mobile devices on information security: A survey of it and security professionals. october 2014.* Retrieved from https://www.checkpoint.com/downloads/product-related/report/check-point-capsule-2014-mobile-security-survey-report.pdf

Chin, E., Felt, A. P., Greenwood, K., & Wagner, D. (2011). Analyzing inter-application communication in android. In *Proceedings of the 9th international conference on mobile systems, applications, and services* (pp. 239–252).

Dykstra, J., & Sherman, A. T. (2012). Acquiring forensic evidence from

 © 2018 ADFSL

infrastructure-as-a-service cloud computing: Exploring and evaluating tools, trust, and techniques. *Digital Investigation*, *9*, S90–S98.

Enck, W., Octeau, D., McDaniel, P., & Chaudhuri, S. (2011). A study of android application security. In *Usenix security symposium* (Vol. 2, p. 2).

Endicott-Popovsky, B., Frincke, D. A., & Taylor, C. A. (2007). A theoretical framework for organizational network forensic readiness. *Journal of Computers*, *2*(3), 1–11.

Felt, A. P., Chin, E., Hanna, S., Song, D., & Wagner, D. (2011). Android permissions demystified. In *Proceedings of the 18th acm conference on computer and communications security* (pp. 627–638).

Fuchs, A. P., Chaudhuri, A., & Foster, J. S. (2009). *Scandroid: Automated security certification of android* (Tech. Rep.).

Gartner. (2013). *Predicts by 2017, half of employers would adopt byod.* Retrieved from `http://www.gartner.com/newsroom/id/2466615`

Gartner. (2015). *6.4 billion connected things will be in use in 2016.* Retrieved from `http://www.gartner.com/newsroom/id/3165317`

Geneiatakis, D., Fovino, I. N., Kounelis, I., & Stirparo, P. (2015). A permission verification approach for android mobile applications. *Computers & Security*, *49*, 192–205.

Grobler, C., & Louwrens, C. (2007). Digital forensic readiness as a component of information security best practice. In *New approaches for security, privacy and trust in complex environments* (pp. 13–24). Springer.

Grover, J. (2013). Android forensics: Automated data collection and reporting from a mobile device. *Digital Investigation*, *10*, S12–S20.

Guido, M., Ondricek, J., Grover, J., Wilburn, D., Nguyen, T., & Hunt, A. (2013). Automated identification of installed malicious android applications. *Digital Investigation*, *10*, S96–S104.

Lars Shmidt Thieme, L. M. (2007). *Multi-label classification.* Retrieved from `https://www.ismll.uni-hildesheim.de/lehre/ml-06w/skript/ml-4up-04-mlabelclassification.pdf`

Li, L., Bartel, A., Klein, J., & Le Traon, Y. (2014). Detecting privacy leaks in android apps.

Miniwatts. (2016). *Internet usage statistics 2016.* Retrieved from `http://www.internetworldstats.com/stats.htm`

Mouton, F., & Venter, H. (2011). A prototype for achieving digital forensic readiness on wireless sensor networks. In *Africon, 2011* (pp. 1–6).

Mylonas, A., Meletiadis, V., Tsoumas, B., Mitrou, L., & Gritzalis, D. (2012). Smartphone forensics: A proactive investigation scheme for evidence acquisition. In *Information security and privacy research* (pp. 249–260). Springer.

Reddy, K., & Venter, H. S. (2013). The architecture of a digital forensic readiness management system. *Computers & Security*, *32*, 73–89.

Rowlingson, R. (2004). A ten step process for forensic readiness. *International Journal of Digital Evidence*, *2*(3), 1–28.

Ruan, K., Carthy, J., Kechadi, T., & Baggili, I. (2013). Cloud forensics definitions and critical criteria for cloud forensic capability: An overview of survey results. *Digital Investigation*, *10*(1), 34–43.

Rumee, S. T. A., & Liu, D. (2015). Droidtest: Testing android applications for leakage of private informa-

tion. In *Information security* (pp. 341–353). Springer.

Shabtai, A., Fledel, Y., & Elovici, Y. (2010). Automated static code analysis for classifying android applications using machine learning. In *Computational intelligence and security (cis), 2010 international conference on* (pp. 329–333).

Statista. (2016a). *Apps downloaded from the apple app store as of sep 2016.* Retrieved from https://www.statista.com/statistics/263794/number-of-downloads-from-the-apple-app-store/

Statista. (2016b). *Number of social network users worldwide from 2010 to 2019 (in billions).* Retrieved from http://www.statista.com/statistics/278414/number-of-worldwide-social-network-users/

Suzanna Schmeelk, A. A. (2014). *Static analysis techniques used in android application security analysis.*

Tan, J. (2001). Forensic readiness,@ stake. *Cambridge, MA.*

Valjarevic, A., & Venter, H. S. (2011). Towards a digital forensic readiness framework for public key infrastructure systems. In *Information security south africa (issa), 2011* (pp. 1–10).

 © 2018 ADFSL