Enhancing Permission Model of Android

Dissertation
Submitted in partial fulfillment of the requirements
for the degree of
Master of Technology

by
Nitin B. Satpal
Roll No: 10305916

Under the guidance of
Prof. D. B. Phatak

Department of Computer Science and Engineering
Indian Institute of Technology Bombay

June 30, 2013
Contents

Abstract 5

1 Introduction 7

2 Architecture of the Android Operating System 8

2.1 Platform ................................. 9
2.2 Core Libraries ......................................................... 9
  2.2.1 Android Runtime .................................................... 9
2.3 Application Framework ............................................. 10
2.4 Application Development on Android ................................ 11
2.5 Application Components ............................................ 11
  2.5.1 Activity ......................................................... 11
  2.5.2 Services ......................................................... 12
  2.5.3 Broadcast Receivers ............................................. 12
  2.5.4 Content Providers ............................................. 13
  2.5.5 Intents ......................................................... 13
2.6 Android Security ................................................... 13

3 Types of Attack ........................................... 15

3.1 Attacks on Smartphones .......................................... 15
3.2 Attacks related to Permission Model ................................ 15
3.3 Attacks not related to Permission Model ........................... 16

4 Literature Survey ........................................ 18

4.1 Section 1 ......................................................... 18
  4.1.1 TISSA ......................................................... 18
  4.1.2 Mockdroid ..................................................... 22
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1.3</td>
<td>Mr. Hide and Dr. Android</td>
<td>24</td>
</tr>
<tr>
<td>4.1.4</td>
<td>TaintDroid</td>
<td>27</td>
</tr>
<tr>
<td>4.1.5</td>
<td>Formal Verification</td>
<td>30</td>
</tr>
<tr>
<td>4.1.6</td>
<td>SORBET</td>
<td>33</td>
</tr>
<tr>
<td>4.2</td>
<td>Section 2</td>
<td>35</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Paranoid</td>
<td>35</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Andromaly</td>
<td>37</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Crowdroid</td>
<td>41</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Permission Based Android Malware Detection</td>
<td>43</td>
</tr>
<tr>
<td>4.2.5</td>
<td>Enhancing Security of Linux-based Android Devices</td>
<td>47</td>
</tr>
<tr>
<td>5</td>
<td>Comparison</td>
<td>54</td>
</tr>
<tr>
<td>6</td>
<td>Problem Statement</td>
<td>57</td>
</tr>
<tr>
<td>7</td>
<td>Implementation</td>
<td>58</td>
</tr>
<tr>
<td>8</td>
<td>Future Work and Conclusion</td>
<td>62</td>
</tr>
</tbody>
</table>
List of Figures

2.1 The Layered Android Architecture ........................................ 8
2.2 Activity lifecycle .................................................................. 12

4.1 The TISSA Architecture ....................................................... 19
4.2 The Privacy Setting Manager ................................................. 20
4.3 Protecting Contacts in TISSA ............................................... 22
4.4 Paper Toss: (a) installation asks for unnecessary permissions; (b) the user is notified via an unobtrusive notification if a mocked permission is used; (c) permissions that are can be modified .................................................. 24
4.5 Mr. Hide Architecture .......................................................... 25
4.6 Dr. Android Architecture ..................................................... 25
4.7 Multi-level approach for performance efficient taint tracking within a common smartphone architecture ......................................................... 28
4.8 TaintDroid architecture within Android .................................. 29
4.9 Entities and relations in the Android permission scheme .......... 31
4.10 Android permissions ........................................................... 31
4.11 Android permissions (refined) .............................................. 31
4.12 Overall Diagram ............................................................... 32
4.13 Paranoid .............................................................................. 36
4.14 Andromoly Application ....................................................... 38
4.15 Work Flow ......................................................................... 40
4.16 Linux User and Kernel Space .............................................. 42
4.17 Android Malware Detection Process .................................... 45
4.18 Intrusion Detection System ................................................. 48
4.19 Intrusion Detection System Architecture ............................... 52
LIST OF FIGURES

7.1 Permission set to 'Trusted' .............................................. 60
7.2 Displaying Original Message ......................................... 60
7.3 permission set to 'Bogus' ............................................. 61
7.4 Displaying Bogus Message ............................................ 61
Acknowledgements

I sincerely thank and express my gratitude towards my guide, Prof. D. B. Phatak, for the constant motivation and guidance he has provided throughout the project. I am extremely grateful to him for spending his valuable time with me to clarify my doubts. Working with him was a great learning experience for me.

Nitin B. Satpal
Abstract

Smartphones are becoming ubiquitous and it has been increase in the number of mobile users who are relying on smartphones to store and handle personal information. However, new applications put user’s personal information at risk. For the protection of user data from third party applications, it is common on smartphone Operating Systems to use permission models to control the Permissions granted to third party applications. The user is given a dialog box with the list of permissions requested by the application before installation. The user can give access to the requested permissions or can deny it and terminate the installation. It is difficult for general users to differentiate between the set of permissions which are potentially harmful and those which are not. Attacker exploit the missing granularity and alterability of most permission model. We have focused on Android permission model and studied various research happened in this field and decided to implement the Open Source version of TISSA along with the integration of Mockdroid in it. Also various third party apps behave as a virus or trojan and affect the performance of the device haphazardly. User once installed the app, give the third party app developer a chance to attack. Attackers nowadays are developing such malicious app to perform various attacks on devices. To protect from such attacks, various solutions are been given. We studied and focused on behavioral based approach of detection of attacks. We have given a comparison of our studied papers and provide the outline of all the studied papers.
Chapter 1

Introduction

The mobile market has expanded at a very massive rate. The year on year growth was logged at 35% at the end of 2010 according to a report. Amongst mobile phones, smartphones have received incredible adoption. The reason of the increased demand for smartphones is the huge availability of applications that can be downloaded and installed easily on smartphones. For example, Google Play of Google contains many Android applications to download. Along with the vendor-provided programs, third-party apps are also present on these marketplaces. The total app downloads have crossed the 25 billion mark.

The increase in the levels of convenience and features of smartphones, causes a significant growth in the number of Smartphone users. For example, phone consists of call log with information about the calls, the user has made and received, an address book that contains the users contact, browsing history, photos etc. As these are all private information, it has to be ensured that they don’t fall in the wrong hands. To protect personal information of user from apps of smartphones which are malicious, a new mode of privacy is needed in smartphones. This new mode mode can modify an apps access to user’s personal information. It will be possible for the user to control the access of personal information to the app in a way that, which information can be accessed and which can not. Further, the user should have run-time control to modify the previously given permission.
Chapter 2

Architecture of the Android Operating System

Though many versions of Android have been released so far, of which the latest is 4.0 dubbed as Ice Cream Sandwich, the core architecture remains the same. It is a layered architecture with the layers consisting of the base linux kernel, libraries, a virtual machine to run applications and an application framework to manage the applications.

Figure 2.1: The Layered Android Architecture
Table 2.1: Linux Kernels used by Android Versions

<table>
<thead>
<tr>
<th>Version</th>
<th>Number</th>
<th>Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cupcake</td>
<td>1.5</td>
<td>2.6.27</td>
</tr>
<tr>
<td>Donut</td>
<td>1.6</td>
<td>2.6.29</td>
</tr>
<tr>
<td>Eclair</td>
<td>2.1</td>
<td>2.6.29</td>
</tr>
<tr>
<td>Froyo</td>
<td>2.2</td>
<td>2.6.32</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

2.1 Platform

The Android Operating System is based on the Linux Kernel. It uses the version v2.6.x of the Kernel tree [14]. The Kernel is an intermediary of the software stack and the hardware. Like conventional Linux systems, it is used for process, memory management etc.

2.2 Core Libraries

The various components within the Android OS use libraries written in C/C++. These libraries are also used by the application framework. These are not the standard libraries present in conventional Linux systems, but a specialized version of a subset of them. For example, there is no glibc present [9].

2.2.1 Android Runtime

A set of Java based libraries for use at runtime are also provided for supporting functionality akin to the core libraries present in the Java programming language [10].

A Java Virtual Machine called Dalvik, modified for use in Android is also provided.

2.2.1.1 Virtual Machine

The Java class files are been converted into .dex format by the Dalvik Virtual Machine. Each instance of Dalvik VM runs one Android application. A separate Linux process is allocated to each instance. Multiple instances can run on one device. Processes isolation, memory management implementation and threading are depend on the Linux Kernel [2]. The Dalvik VM is
Table 2.2: Percentage Memory saved by Dalvik VM (treating Uncompressed jar as 100%)

<table>
<thead>
<tr>
<th>File</th>
<th>Compressed .jar</th>
<th>Uncompressed .dex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common System Library</td>
<td>50</td>
<td>48</td>
</tr>
<tr>
<td>Web Browser App</td>
<td>49</td>
<td>44</td>
</tr>
<tr>
<td>Alarm Clock App</td>
<td>52</td>
<td>44</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

designed, so as to run optimally on a machine with slow CPU, less RAM, and without swap space OS, while having the additional power constraint of running on the battery [2]. It is optimized to weed out redundancy that a regular .jar file has, while integrating every existing Java class file. It does so, by efficiently linking constant information as shown [2]. Thus, memory saved via minimal repetition, implicit typing and labeling. Level of memory saved is tabulated as shown in table 2.2.

Dalvik VM is register-based. This helps in avoiding unnecessary memory access and getting more meaning per instruction. As a result, the same work is being done with 30% fewer instructions and 35% fewer code units [2].

2.3 Application Framework

Application framework layer defines how applications are built and how they behave. The Android application framework includes [1]:

- User interface for application which is built by views including lists, grids, text boxes, buttons, and embedded web browser
- A set of Content Providers which enable applications to share their data and to access data from other applications
- Access to resources is provided by a Resource Manager
- A Notification Manager enables the display of custom alerts
- An Activity Manager provides the common navigation stack for all application and also manage application life cycle
2.4 Application Development on Android

Java programming language is used to develop the applications in Android. Compilation of
code is done by various Android tools with the help of other information in an archive with
an .apk suffix. All the code in a single .apk file is considered to be one application and is the
file that devices running on Android, use to install the application [1]. One can publish the
application over the Android Market so that it can be distributed to other users as well. An
application can only use the resources, that it has permissions to do so.

2.5 Application Components

Android application is made up of different Application components. These components are
the different entry points for the system. There is no strict or set policy defined governing
where a component can actually enter the system. There isn’t even a main() function defined
for application processes. For user, each and every component doesn’t serve as entry point.
Components can be inter-dependent, but every component has a specific role and has its own
entity. The application’s overall behavior is depend on these building blocks i.e components.
Different types of components are as follows:

2.5.1 Activity

An activity is simply a task from the overall application with a user interface for input. It
can be the Search bar in an Address Book or the settings pane in a game. Each activity,
though independent, together form the whole application. It is implemented as a subclass of
the Activity class. One activity can be used to call another, if allowed. The activity component
has a specific lifecycle as shown in figure 2.2 below:[1]

Broadly put, three types of nested loops are created due to the lifecycle each having a specific
role.

- Entire lifetime: between onCreate() and onDestroy()
  Global setup should be performed. E.g. Creation and connection of sockets as well as
  their destruction for file transfer.

- Visible lifetime: between onStart() and onStop()
  Activity is displayed on the screen. E.g. Modifying the progress bar for the file transfer.
2.5.2 Services

A service is a long-running component which doesn’t have an interface with the user, but does so with an activity. An activity can take events from a service or it can start or stop it. For example, the system clock can be a service for an Alarm Clock app.

2.5.3 Broadcast Receivers

System wide broadcast is the responsibility of the Broadcast Receivers. They response as soon as any information is being broadcasted. E.g: A broadcast announcing low battery. Broadcasts can also be initiated by applications. There is no user interface for Broadcast receivers,
CHAPTER 2. ARCHITECTURE OF THE ANDROID OPERATING SYSTEM

but a status bar notification can be created to alert the user. Usually, broadcast receiver is implemented as a ‘gateway’ to other components[1].

2.5.4 Content Providers

Data is shared among application using Content Providers. The data could be in file system, SQLite database or any other persistent storage location, but the application must have access to that storage location. The data format supported is defined by the Content providers. Content Provider define a set of methods using which the application can query or modify data. But there is not direct communication between applications and Content Provider. All the communication between them are done by an intermediary, Content Resolver.

2.5.5 Intents

Intents are asynchronous messages. Intents activate activities, services and broadcast receivers. They can be used to connect individual components from the same or different application. They may define the action to be performed, be the event in case of an activity or service, or simply make an announcement in the case of a broadcast.

All non code resources are defined in XML files and those are separate from the source code. A unique integer ID is defined by SDK build tools for all the resources which are included in an Android project. By using this id, a resource is referred from application code or from other resources. Hence, it become easy to update different characteristics of an application without the code being modified. Sets of alternative resources are provided which enables developers to optimize the application[1].

2.6 Android Security

In Android, each application has a distinct system identity i.e Linux user ID and group Id. Some Parts of the system are separated into distinct identities. Application are isolated from each other and from the system in Linux. Permission mechanism is used to give additional finer-grained security features. Using this mechanism restrictions on the specific operations can be enforced that a particular process can perform.

The Android system defines four protection levels, level 0 to 3, for permissions.
Level-zero (0) permissions are so called normal permissions which pose a low-risk factor and typically only affect the applications scope. Level-zero permissions are granted by the system automatically, without explicit approval of the user. Optionally, the user can request to be notified of the permission request, prior to the installation of the application.

Level-one (1) permissions are higher-risk permissions, e.g. allow costly access to services such as initiating phone calls or access to the devices sensors, the Internet, or sensitive user data. An interesting permission on Level-one is the permission to read the device’s log files. Prior to the installation, the package installer displays the set of requested dangerous permissions to the user, which decides to either grant or deny the set of permissions and if the user gives his consent to all of the requested permissions, the application can successfully be installed.

Level-two (2) permissions are only granted if the application that is being installed, is signed with the private key corresponding to the same certificate as the application that originally defined this permission. These so called signature permissions can be used by developers, e.g., to share information between their own application, while preventing applications of other developers to gain access to this information. So even if the user would consent, a signature permission cannot be granted to applications signed with the private key corresponding to another certificate.

Level-three (3) permissions can be granted by the system to applications that are contained in the systems image, or by applications that have been signed with the same certificate as the system image. Permissions of this highest category are reserved for a handset of manufacturers and OS developers. Representatives of this category are the permissions to install new application (packages) or to change security settings.
Chapter 3

Types of Attack

3.1 Attacks on Smartphones

For the sake of our study, we have distinguished the attacks in two types as follows;

- Attacks related to Permission-Model (Data-stealing attacks)
- Attacks not related to Permission Model

3.2 Attacks related to Permission Model

In such type of attacks, an attacker made the third party app and upload it on the Google Play. Android users download the app and install it on their smartphone. The attacker ask different permissions at the time of installation. As the Android Permission Model is coarse-grained, user is bound to give access to all the requested permissions in order to successfully install the app.

Now, while the user uses the app, in the back-end the app can misuse the permission granted to it to steal the private data of the user. Such attack exploits the vulnerability in the design of Android Permission Model.

We have studied various papers in Stage 1 and proposed the solution which is being discussed in TISSA to make the Permission Model more fine-grained. We implemented the proposed model for six different permissions and also provide the run-time control to the user which was not the part of original model proposed by TISSA.
CHAPTER 3. TYPES OF ATTACK

The literature survey for such types of attacks is discussed in section 4.1 and the implementation detail of our model is given in section 7.

3.3 Attacks not related to Permission Model

Such types of attacks consists of Performance degradation attacks, Denial of Service attacks and other such attacks.

Various anti-virus and other attack detection mechanisms are been proposed to get rid of such attacks. We compared the performance of such tools or softwares in order to check the impact of running such softwares on battery power and time elapse.

We Studied and Performed experiments with three anti-virus (attack- detection) softwares namely

- AVG anti-virus
- Lookout anti-virus
- Bitdefender anti-virus

Following is the table of the performance comparison; Experiments were performed on Samsung Galaxy Y.

<table>
<thead>
<tr>
<th></th>
<th>No. of Apps</th>
<th>Time</th>
<th>Battery Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVG</td>
<td>140</td>
<td>27 mins</td>
<td>21.00%</td>
</tr>
<tr>
<td>Lookout</td>
<td>140</td>
<td>11 mins</td>
<td>35.00%</td>
</tr>
<tr>
<td>Bitfinder</td>
<td>140</td>
<td>18 mins</td>
<td>29.00%</td>
</tr>
</tbody>
</table>

Following are the advantages and disadvantages of such tools

Advantages

- Easy to install on most of the phones
- Can detect well known virus and trojans etc.
Disadvantages

- Can’t detect zero-day attacks.

- Zero-day attack \[18\]:
  It is an attack that exploits a previously unknown vulnerability in a computer application, meaning that the attack occurs on “day zero” of the awareness of the vulnerability.

- Signature-Based \[17\]:
  In such approach, new virus cannot be detected till we get the signature of that, which may need 48 hours or so atleast. So the device will be vulnerable to the attack atleast for 2 days.

- Can detect virus, trojans. But not able to protect the stealing of data.

- Slow and battery consuming

In stage two, along with the implementation of TISSA model, we also studied various papers on improving the performance of smartphones while detecting various attacks of the current era. The detailed literature survey is given in section 4.2. We compare different approaches and briefly concluded our review on such approaches in section 8.
Chapter 4

Literature Survey

4.1 Section 1

4.1.1 TISSA

The goal of the TISSA [19] is to design a model which can prevent private information leakage by untrusted smartphone applications effectively and efficiently. TISSA has the following design requirements, in order to provide the protection to the private information with a considerable system performance, better user experience, and application compatibility.

- Lightweight Protection: The proposed mechanism should be efficient in terms of both memory and energy. User experience should not be affected due to Performance overhead.

- Application Transparency: The model may not change the APIs provided by the default Android framework and should maintain the compatibility of the existing Android application.

- Small Footprint: The necessary changes to the Android framework should be minimum.

As shown in the figure, the model is made up of three components:

- Privacy setting content provider: It is the Policy Decision Point in the model and is responsible for the privacy settings of untrusted applications. Using an API, it queries the current privacy setting for an installed application.

- Privacy setting manager: Its role is of Policy Administration in TISSA. This component is used to update and manage the privacy settings of installed applications by the mobile user.
Privacy-aware components: These components work as Policy Enforcement Points. Content Providers or services fall in this category. These components are responsible for the access of users personal information like contacts, call log, location, and phone identity. These components receive request from applications to access private data, and further privacy setting content provider is being queried, for the permissions given to the application, and they respond accordingly.

Single instance of PDP and PAP is present, while multiple instances of PEPs exist. PEPs are integrated with individual content providers or services in the Android framework.

The workflow of TISSA is explained as follows:

To read private data, application sends a request to the corresponding provider. A query is fired from Content Provider to privacy setting content provider and holds the request of the application, until it gets the response from privacy setting content provider. The privacy setting content provider, fetches the user specifications on privacy settings for a specific application, by querying to its internal policy database. The result is then returned to the content provider. The provider serves the access request, if reading operation is permitted, and returns the normal results to the application. The privacy setting content provider may indicate possible ways to handle it, if the reading operation is not permitted.

In TISSA, three options are supported; empty, anonymized and bogus. In empty option, the content provider returns an empty result to the application which indicates the non-presence
of the requested information. As the name suggests, Anonymized version provides anonymized version of the original data. In this case, the application continues to run, but without having the requested personal privacy information. Bogus option provides the fake result to the requested application. The options can be interpreted differently for different applications. The applications will behave accordingly, based on the returned results. Hence, the mobile users need to be aware of this different behavior of the same application under the different privacy setting.

TISSA provides three levels of granularity. The policy defines whether a particular application can be completely trusted in the first level. If yes, it is given all the requested accesses through the normal Android permission mechanism. In other case, TISSA provides the second level of policy specification, where one particular setting can be specified for each type of personal information the user wants to protect. It is possible that certain application may access one type of personal information for its legitimate functionalities and should be denied for other personal information. The third level of policy specification specifies empty, anonymized, and bogus options as per the need. For example, an anonymized version of call log can be given to the application and it can simply return a bogus value to the phone identity.

Figure 4.2: The Privacy Setting Manager
In the figures given above, two different activities of Privacy Setting Manager of the smartphone is shown. In the first one, all the installed applications are listed, while in the other one, the privacy setting of the application Yellow Pages is shown. The first activity is a PrivacySettingManagerActivity which in turn starts the other activity called AppPrivacySettingActivity. This activity sends the query to the privacy setting content provider to obtain the current privacy setting and displays the results to the user. It also allows the user to customize or adjust the current settings.

Following is the example of how TISSA protects the contacts.

Figure 4.3 shows the flow chart for the contact information that is being accessed. When an application makes a query to the content provider for contacts, the request is accessed by the resolver, which checks the permission given to the application. If the permission is granted, the request is dispatched to the contacts content provider. In privacy mode, the contact content provider sends the query to privacy setting content provider, to check if the requested query can access the contacts data. If yes, then it queries its own database and returns the authentic contacts to the application. If the application is not trusted to access the requested data, the contact content providers will send empty contact record, anonymized version of the contact records or bogus i.e fake contact information. In the meantime, the application will behave differently, and the user will experience an unexpected or abnormal behavior of the given application.
4.1.2 Mockdroid

Mockdroid is the model which mocks the data given to the application, if permission is not given to access that data [3]. Mockdroid supports in mocking the resources below:

- Coarse and fine grained location: It simulates lack of information on location if one mocks read requests
- Internet: It simulates that available wireless networks are not available if one mocks the internet
- SMS/MMS, calendar, and contacts: It simulates a full database if write requests are mocked and an empty database if read requests are mocked, leading them both to fail
- Device ID: It simulates the return of a fake constant value if we mock read requests
- Broadcast Intents: A broadcast intent doesn’t get sent if one mocks the permission to read from a package. Similarly one doesn’t receive it, if the permission to write is mocked. This is used by Mockdroid to avoid the receipt of incoming SMS/MMS message from an application
During installation of a new app from the market, the user is displayed all permissions, however dangerous they may be and is asked to allow them. The installation fails if user denies the permission, otherwise the set of permissions that have been allowed are stored in a data structure still in memory which is then copied to disk. Each API call requiring a permission checks if the proper permission has been allowed to the package calling the API. If an attempt is made by an app to use an API call without proper permission, an API throws the runtime exception to package.

The permission check when a dangerous API call starts has been modified by Mockdroid. The exceptions generated when a permission was not requested during installation is thrown just like Android does. If a permission was allowed during installation, then its checked if it was mocked. Each permission’s state is maintained so that for a resource, one can prevent access to one app and enable it for another.

In Mockdroid the package manager has had some modifications and the permission set has been duplicated so as to maintain a real and mocked version for each permission. All the permissions that are requested are allowed and no one is mocked yet. Mockdroid uses an additional Unix group, called mock, which has been added to the Android OS. It uses the inotify kernel service to keep an eye on the files in the mocked permission directory and see if any have changed. The update of the changes in in-memory cache of mocked permission as and when the file contents changed.

Permissions related to the internet are supported in a different manner from other Android API calls which are dangerous because the internet is implemented via API calls from the kernel and the virtual machine. Standard Android checks whether a process wanting access the internet belongs to the inet group. If an application has granted the internet permission, Android Activity Manager adds the process of the application to the inet group.

Following is the example of a gaming application Paper Toss using Mockdroid:

Figure displays the list of permissions which the application needs for installation. All the information seems to be unnecessary, as there is no need for internet permission, phone state and identity, coarse and fine grained location. By mocking the permission, all these private data can be saved from leaking. Now, suppose the user wants to save the highest score to an online
Figure 4.4: Paper Toss: (a) installation asks for unnecessary permissions; (b) the user is notified via an unobtrusive notification if a mocked permission is used; (c) permissions that are can be modified

table having the highest scores, it wont be possible, as internet permission are mocked. Figure (b) shows the scenario in which Mocker application shows the user that internet permissions are mocked and access cannot be given. The user can enable the internet access through the notification as shown (figure c).

It is absolutely possible if the user again wants to mock the internet permission after submitting the score.

4.1.3 Mr. Hide and Dr. Android

The current Android Platform allows coarse grained permission. The system called Mr. Hide and Dr. Android replaces this coarse grained permissions to fine grained [6]. Hence, it lowers the need of privilege levels and decreases the potential threat from vulnerabilities and malicious applications. It comprises of two components; Mr. Hide and Dr. Android. Mr. Hide is a service which protects sensitive device capabilities with fine grained permissions that the service dynamically enforces. For an example, Mr. Hide includes an API to access the internet but it protects the API with a new permission InternetURL(d), which grants permission to access domain d only, on the internet. Dr. Android on the other hand is a tool that modifies
existing Android application package files so that Mr. Hide can be used. Dr. Android takes an apk and the list of finer-grained permissions to retrofits as an input. Dr. Android and Mr. Hide together enforces applications to use their system for accessing the private data of users. In short, this system protects dangerous permission from getting misused. Dr. Android can change the permission set of an apk; rename platform API references in bytecode to refer to their Mr. Hide counterparts; append the Mr. Hide adapter layer to the code in the apk file; and insert code to bind to the Mr. Hide service among other similar transformations. Adapter layer is an drop-in replacement for sensitive platform APIs as well as common third party libraries in Mr. Hide. It takes care of all necessary communications with Mr. Hide service.

Following is the architecture of Mr. Hide and Dr. Android:

Figure 4.5: Mr. Hide Architecture

Figure 4.6: Dr. Android Architecture
Figure (a) shows the architecture of the Mr. Hide. It contains two main components, Mr. Hide service and hidelib. Mr. Hide service runs in a separate process, while hidelib is a drop-in replacement for sensitive Android APIs that manages all interprocess communication with the Mr. Hide services. hidelib is nothing but the adapter layer which is explained earlier and is appended to the original applications apk.

Figure (b) shows the architecture of the Dr. Android. It takes apk file as an input to the apktool and decompress it to its constituent files and directories. These constituents are .dex file, manifest.xml file and other resources (.xml) as shown in figure (b). Dr. Android modifies the classes.dex file which contain Dalvik bytecode for the app and enforces the application to use the Mr. Hide service. For this, it concatenates hidelib.dex, an adapter layer to connect to Mr. Hide service which is running in a separate process, to the output of the classes.dex file. Dr. Android also modifies the list of permissions in AndroidManifest.xml file which is the applications manifest file and contains the permissions requested at application installation time. In some cases, some other files also need to be modified which is shown as other resources in the figure. All these modified files are then repackaged using apktool to produce a transformed apk.

Following is the example of how this system can be used to improve the security and privacy of an application - Horoscope which is an popular free android application. This application allows user to check the horoscope for different astrological symbols for the current day, next day, current week, etc. This application requests several permissions which are classified as dangerous permissions by Android. It include access to the internet, read the phone state, access fine i.e GPS-based and coarse i.e Network-based location information and write system settings. By using Mr. Hide and Dr. Android less privilege policy can be enforced to the application for an access-control.

- To reduce the scope of the internet access, InternetURL(d) permission can be used provided by Mr. Hide in which the access of the internet will be very less and will be allowed only for the given domain d.

- The user can also use AdsPrivate and AdsGeo to ensure that request to one commonly used ad server AdMob, do not leak sensitive information. It can be leaked by the applications access to the internet in the absence of this system.
• By using Mr. Hides UniqueID permission and associate library for this case, false device ID can be send.

• If user wants to protect the detailed location information, Mr. Hides LocationBlock permission can be used. It provides location information that is only accurate up to around 150 metres.

### 4.1.4 TaintDroid

It is a detective measure model [4]. Using this model, it can be tracked, where the private information data is going from smartphone when user is using a certain application. This model will detect the information flow of the user while using a particular application and will report whether the data sent is going to any third party server or the application which the user is using is not leaking user’s private data. But it is not possible using this model to prevent the leakage of privacy information. The challenges in monitoring private information via the network on smartphones as follows:

• Smartphones are limited in resources: Resource limitation of smartphone make the use of resource-heavy information tracking systems such as Panorama unusable

• Access to private information is stored in Third-Party applications. Hence the system must know which type of information it is and this requires more computation

• Private information is based on context is dynamic and it is difficult to identify it. An example is geographic location which is a pair of floating point number which change frequently

• Applications can involve information flow between apps, hence it is not a good idea to limit monitoring on a single application

TaintDroid uses taint analysis in a dynamic way to monitor private information on smartphones. It tracks how data that is labeled impacts other data that might give an idea on the original private information. The impacted data is identified before it leaves the system.
There are four levels of tracking:

- Variable
- Message
- Method
- File

Figure 4.7: Multi-level approach for performance efficient taint tracking within a common smartphone architecture

First, it instructs the VM interpreter to track variables in untrusted application code. By tracking variables provided by the interpreter, taint explosion can be avoided observed in c86 instruction set. Also, by monitoring variables, taint markings are used only for data. Message level tracking is a type of interprocess tracking i.e tracking of messages which are exchanged between the applications. It minimizes the overhead from inter-process communication while extending the analysis to the entire system. Method-level tracking is used for system-provided libraries. In this level of tracking, native code is run without any instruction and patches the propagation of taints in return. Finally, TaintDroid uses file tracking to ensure the persistence retains its taint markings. TaintDroid’s architecture within Android is as shown in the following figure.

The marking (1) shows the tainting of information in a trusted application with sufficient context. (2) shows that the taint interface invokes a native method which in turn interacts with Android’s VM interpreter and store specified markings of taints in a virtual map. (3) shows the propagation of the taint tags by the Dalvik VM according to the data flow rules. Every instance of an interpreter propagates taint tags at the same time. When trusted application
uses such information which is tainted in an IPC transaction, the modified binder library (4) ensures the combined markings of the taints of all contained data is seen in the parcel.

How to store the taint tags in turn affects performance and memory overhead. Dynamic taint tracking systems store tags for every byte or word of data. Tags that are frequently tainted use in non adjacent shadow memory for storage. They also use tag maps. Figure shows how the taint propagate.

There are two exceptions, the logic of taint propagation usually include that during lookup the taint tag of an array index handle the table for translations. For example, in a table that translates characters from lowercase to uppercase , if a tainted value b is used as an index to an array, the corresponding B value should be tainted even though its value in the memory is not tainted. So the logic for a retrieval operations uses both taints - array and array index. When object references are contained in an array, the taint in the index tag is propagated to the object reference and not to the object value. Hence, model includes the object reference taint tag for retrieval rule.

Native code is unmonitored in TaintDroid. In java environment, there are two postconditions for accurate taint tracking; All accessed external variables are given tags corresponding to the data flow rules. The data flow rules assign return values a taint accordingly.

Using manual instrumentation, heuristics, and method profiles, depending on requirements and depending upon the situation, TaintDroid achieves these conditions. Using direct inter-
preted code TaintDroid can call internal VM methods by passing pointers to a 32-bit register arguments and a return value. For taint propagation TaintDroid manually inspects and patches them as internal VM methods are relatively small in number and infrequently added between versions. While for JNI methods, the JNI call bridge is used to invoke the methods and it also goes through arguments and assigns a return value. Hence, TaintDroid provides taint propagation for all the JNI methods.

To propagate the taint tags while applications exchange data between them, message level taint marking is used. A Taint tag of message is nothing but the upper level of taint markings of variables in the message. All items in the message pertaining to data share the same taint tag which might lead false positive. A term false positive means considering some variable or data item as tainted though it was not. Taint tags should not be lose while data is stored in a file. One taint tag per file is stored by it. This tag gets updated on file write and gets propagated on file read. As one taint tag is stored per file, false positive can occur which reduces taint markings to coarse grained for information databases. Extended attributes in the file system are used to store file taint tags. This is done by implementing attribute support for the Android file system and formatting the external storage card with ext2 file system.

4.1.5 Formal Verification

This model gives a formal model for analysis of the permission authorization and enforcement in Android \[15\]. For this, the model use the Entity-Relationship and state diagrams. Model shows various ER diagram in order to explain the concepts of Permission scheme used in Android framework. The first ER model is as per the applications, components and permissions used in the Android framework.

One or more components together constitute an application and all the components are introduced in the system as applications are installed. The COMPOSE relation shows that the application is composed of one or more components and every component composes one application. An application doesn’t has to have permissions, and as the application is installed, the permissions are introduced into the system. The DECLARE relation shows permissions that are optionally declared by the application. But each permission is declared by an application. An example of optional N-to-M relationships are USE, AENFORCE and CENFORCE. An application can use some of these permissions. Enforcement can also be done by an application.
or a component and vice-versa.

Following is the figure of ER diagram considering Permissions, Objects and Operations;

Figure 4.10: Android permissions

Here only one more relation is more or extra added and that is of EXWITH which shows operations are executeWith these permissions and permissions are enforced to operations. The relation in above figure can be refined as shown below;

Figure 4.11: Android permissions (refined)
Finally the overall diagram of the Android Permission scheme is as shown in the following figure:

![Overall Diagram](image)

Figure 4.12: Overall Diagram

The overall diagram also contains certs and plev as new terms. CERTS is the set of certificates used to sign the applications. signedWith is the relationship between application and certs which shows a particular application is signed with a particular certificate. PLEV is a set of protection levels i.e NORMAL, DANGEROUS, SIGN, SIGNORSYST given to the permissions. signProtected returns true when the permission is protected with a protection level SIGN or SIGNORSYS else return false.

One component can perform operations on another. These operations form interactions between components. scmp and ocmp are the two performers and op is the operation which is performed by scmp on ocmp. It can be denoted as a tuple (scmp, ocmp, op). checkAccess is used to check whether scmp has all the permissions that ocmp enforces because some operations have protected because of permissions. The checkAccess returns false unless it owns all the permissions. The set of permissions that ocmp enforces can be calculated as follows:

\[
\{ p : \text{PERMS} \land p \in \text{enforces(ocmp)} \land p \in \text{executedWith(op)} \}
\]

If the following condition is satisfied, only then it returns true though the set of permissions that scmp uses can be obtained by using composes() and uses().

\[
\forall p : \text{PERMS}, p \in \text{enforces(ocmp)} \land p \in \text{executedWith(op)} \Rightarrow p \in \text{uses(composes(scmp))}
\]

STATE is the set of states that the system that runs the Android permission scheme has.
STATE is a tuple of \( (A, C, P, AC, DP, UP, EP, R, X) \), where \( A \subseteq APPS \), \( C \subseteq COMPS \), \( P \subseteq PERMS \), \( AC \subseteq A \times C \), \( DP \subseteq A \times P \), \( UP \subseteq A \times P \), \( EP \subseteq C \times P \), \( R \subseteq C \), \( X \subseteq C \times C \times OPS \).

The described mapping can be used in state transition scheme also with some modifications. For example, the composes of the mapping can be extended to

\[(s : STATE, cmp : COMPS) \rightarrow APPS\]

Following are the operations which can change the state of the system and also the abstract operations using these primitive administrative operations, state and elements to track whether the given operation causes a system to go to the different state or the state transition is occurring or not.

Also the state validity is checked using the same types of equations. Some of the examples are given as follows:

- If an application performs an operation relating to security on another application it means that the application has proper permissions. This statement is obtained by modifying Google’s Security and Permissions statement and can be converted into the following.

- The permissions in the manifest file are the ones that the file uses. Though for signature-level permission, signatures of the one who requests and one who declares the permission need to be compared

- The subject component of an ongoing access must be in the running state. An installed application must have a running component

All the statements are converted into security conditions to check whether the system is following a series of state transitions which makes it lead to an invalid state.

### 4.1.6 SORBET

The model named Sorbet generalizes Android-style permissions and instantiate the current permission system of Android \([6]\). It has been shown with the help of the model that Android
currently does not obey some of the desired security properties. Sorbet describe a set of improvements to Android permission system which support coarse-grained information flow and privilege-escalation policies.

As the figure also lists run-time components, it is important to distinguish between run-time instances from run-time components and run-time instances from each other. A static component C may have multiple run-time instances iC. All the terms in the figure are self-explanatory, but the details of the terms can be obtained from section 2.2 of [6].

Model proposed the following desired security properties;

- Local callee property: If a component A is called by another component B, then As guard ckCallee evaluates to true.

- Local caller property: If a component A calls another component B, then As guard ckCaller evaluates to true.

- Delegation: A component A has a permission P if A owns P, or there is a delegation chain from a component B to A such that A satisfies the scope and lifetime constraints imposed by every component on the chain, and that every component on the chain also has P.

  - A owns P or A is granted P by the user; or
  - There exists a delegation chain that ends with A s.t. for all B in the chain (where B = A):
    - B has P, and A satisfies the scope and lifetime constraints imposed by B;
    - and P has not been revoked from A.

- Revocation: If A revokes P from B, then there is a delegation chain from A to B, or A owns P.

- Privilege escalation: Given any component B protected by permission P, and any component A that does not have that permission, if SAB is a system that contains A and B (and other components), and SB is the same system without A, then a call chain ending with B exists in SAB if and only if it exists in SB. Additional call chains ending with B may exist in SAB if explicitly allowed by policy.

34
• Information flow: Given any component B protected by permission P, and any component A that does not have that permission, if SAB is a system that contains A and B (and other components), and SB is the same system without A, then a call chain ending with B exists in SAB if and only if it exists in SB. Additional call chains ending with B may exist in SAB if explicitly allowed by policy. Given an undesired information flow from a component A guarded by P1 to a component B guarded by P2, a call chain that ends with B exists in a system with A if and only if the same call chain exists in a system without A. Additional call chains ending with B may exist in the system with A only if explicitly allowed by policy.

All the above properties have been obtained in Sorbet model, while in Android permission scheme, only first two properties hold.

4.2 Section 2

4.2.1 Paranoid

4.2.1.1 Introduction

Paranoid [10] is a proposed model to work on the detection of attacks on smartphones having Android. Attack detection mechanisms are mostly degrade the performance of the phone in terms of battery power and processing speed. The main reason behind this, is the heavy detection mechanisms which used ample amount of battery and CPU power. Hence the model propose to have all the attack detection mechanisms to be performed on cloud, where the exact replica of the phone will be created.

The motivation behind this is that server have no such constraints which are there on phones. Multiple detection techniques can be applied simultaneously on the remote server / cloud to get the better results. A different security model to devolve attack detection from phone is suggested by Paranoid.

4.2.1.2 How It Works

The model proposed one tracer application which will be working on phone, while one replayer application will work on remote server. On server side, the replica of the phone will be run on the emulator. For the connection to internet and temporary storage of data, a proxy is present.
All data that is being transferred from kernel space to user space through system calls, is recorded by the tracer. The replacer replays the system calls to the replica by using the values recorded by tracer.

The limitation with this model is that, only process is being replayed and not the kernel, hence the attack on the kernel level cannot be detected. But in general, to attack on the kernel, first the user level process need to be compromised. Hence Kernel is also getting protected indirectly.

Following is the figure to give the brief idea how the overall model behaves;

![Figure 4.13: Paranoid](image)

### 4.2.1.3 When the Phone will be synchronized?

- **Loose Synchronization**: It synchronized phone with server only when it is connected to internet.
- **More Loose Synchronization**: It synchronized phone with server only when it is connected for recharging.

### 4.2.1.4 Detection Techniques

- Dynamic analysis in the emulator
- AV products in the cloud (File scanning / on access file scanning)
• Memory Scanners

• System call anomaly detection

4.2.1.5 Recording and Replaying

• Init start the tracer process
  The process that are to be traced are launched by Init using execution stub. Tracer initializes FIFO for the processes to be traced

• A level of indirection has been added in the starting of other processes i.e. stub

• A process created using ‘fork’ and instead of advancing to ‘exec’, it contact stub, which added the pid of the process to FIFO

• Tracer then attaches to the process one by one and after tracing each process, contacts the stub and the stub make the exec call

4.2.2 Andromaly

4.2.2.1 Introduction

Andromaly propose a lightweight Malware Detection system for Android-based mobile devices. It assists user in detecting suspicious activities on their handsets.

Real-time monitoring, collection, preprocessing and analysis of various system metrics is the basis of Andromaly. The system metrics varies from CPU consumption to number of sent packets through the WiFi. It also monitors number of running application and battery level.

Various detection units or processors are used to analysis the above metrics. For the detection of malicious behavior, each processor employ its own expertise, and generate threat assessment accordingly. An integrated alert is produced by weighing the pending threat assessments. A different weighing criteria is there for Virus threatening assessments and worm threatening assessments. After weighing phase, notification is generated to the user.

4.2.2.2 How it works

Figure shows the architecture of the Andromaly application;
The components of the architecture are broadly classified in the following categories:

- **Feature Extractor**: It communicates Linux Kernel and application framework layer and collects feature metrics. The role of Feature Manager is to collect new feature measurements after every predefined time interval. It triggers Feature Extractor to request the features.

- **Processor**: It works as the analysis and detection unit. The feature vector provided by the Main Service are analyzed by the Processor and output is given to the Threat Weighting Unit. The output of the processor is the Threat Assessments.

- **Threat Weighting Unit**: All the active Processors gives the analysis result to the TWU. The final result about device’s infection level is derived by TWU after applying algorithms like Majority voting, Distribution Summation etc.

- **Alert Manager**: It apply some smoothing function after receiving the final markings by TWU. It avoid an instantaneous false alarms and provide a more persistent alert.
Main Service:
It plays the central role in the model. Feature collection, malware detection and alert process are synchronized by Main Service. It request the new sample of features. After receiving, the signals are send to Processors. After all processing, Alert Manager give final recommendation back to the Main Service. Following are the components of Main Service

- Loggers: The Loggers provide logging options for debugging, calibration and experimentation with detection algorithms.

- Configuration Manager: The configuration of the application is managed by Configuration Manager. It includes different configurations like active processors, active feature extractors, alert threshold, active loggers, sampling temporal interval, detection mode configuration, etc.

- Alert Handler: A dispatched alert causes Alert Handler active and it triggers the required action.

- Processor Manager: Registration and de-registration of processors is handled by Process Manager. It is also responsible for activating and deactivating processors.

- Operation Mode Manager: As the name suggests, it changes the mode of application from one operation to another which in turn causes activation or deactivation of Processors and Feature Extractors

4.2.2.3 Work Flow
Following is the diagram showing the work flow of the model:
4.2.2.4 Detection Method

The classification techniques of the Machine Learning are used in Malware Detection System. Various events and features obtained are continuously monitored. Standard Machine Learning classifiers are then applied and collected observations are classified as either normal (benign) or abnormal (malicious).

4.2.2.5 Feature Selection

Irrelevant and redundant features can present several problems such as misleading the learning algorithm, increasing complexity etc. Hence, to use malware detector more efficiently and with a faster detection cycle, fine feature selection is important.

Filter approach is used to select the feature. The three methods which are applied to the datasets are; Chi-Square, Fisher Score and Information Gain. Three different configurations: 10, 20 and 50 features are applied to each algorithm. These feature are among highest ranked features out of the 88 features.
4.2.3 Crowdroid

4.2.3.1 Introduction

Crowdroid [4] is a machine learning-based framework that recognizes Trojan-like malware on Android smartphones, by analyzing the number of times each system call has been issued by an application during the execution of an action that requires user interaction. A genuine application differs from its trojanized version, since it issues different types and a different number of system calls.

Crowdroid builds a vector of ‘m’ features (the Android system calls). A lightweight client Crowdroid is developed. This client can be downloaded and installed from Google’s Market. The Linux Kernel system calls are monitored by this client and are send to a centralized server. Non-personal behavior related data from user can be used for clustering and further processing. Remote server will create one system call vector for every interaction of user with the application. In this way the dataset of each application will be created.

Partition clustering algorithm is used to cluster each dataset. In Linux, a program requests a service from the operating systems kernel, by invoking a system call. Useful functions are provided to application programs by system call. These functions includes network, file, or process related operations.

4.2.3.2 How it Works

when an application makes a request to the Operating System from user space, the call goes in the direction from Glibc library, System Call Interface, Kernel and finally to Hardware. The call is interpreted by Glibc and mode of CPU is switched to Kernel mode. A request is made to the hardware by Kernel and the requested information is send in the inverse process by the application, in the user space.

The model capture the system call via system call interface through Kernel by using ‘strace’. An output file with all events initiated by the Android application is generated by the model. Useful information about files, time stamps of execution and the count of each system call number that are executed by the application are contained in the file. The model is using the feature “number of times each system call executed".
Following are some important components of the model;

- **Data acquisition**
  
The role of this component is to obtain an application data from users. For this the Crowdroid application is used. This application data is composed of information of device, list of installed applications and the result of monitoring applications.

- **Data manipulation**
  
  It manage and parse all the information which is collected from Android users. It collects information from ‘strace’ output files. It extracts this information and analyzes. Devices basic information are stored in a central database, and system call traces are processed to produce the feature vectors that will be used for clustering.

- **Malware analysis and detection**
  
The feature vectors obtained from the previous phase are analyzed and clustered to create the normality model. Then the anomalous behavior in Android applications is detected, using K-means clustering applied on system call count feature vectors.
Table 4.1: Self-written malware result

<table>
<thead>
<tr>
<th>App.</th>
<th>Interactions</th>
<th>Clustering result</th>
<th>Detection rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Good App</td>
<td>Malware App</td>
<td>Good Clustered</td>
</tr>
<tr>
<td>Calculator</td>
<td>50</td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>Countdown</td>
<td>50</td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>Money Converter</td>
<td>50</td>
<td>10</td>
<td>50</td>
</tr>
</tbody>
</table>

Table 4.2: Result for Monkey Jump 2 Application Behavior Clustering

<table>
<thead>
<tr>
<th>App.</th>
<th>Interactions</th>
<th>Clustering result</th>
<th>Detection rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Good App</td>
<td>Malware App</td>
<td>Good Clustered</td>
</tr>
<tr>
<td>Streamy Window</td>
<td>6</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Monkey Jump2</td>
<td>15</td>
<td>5</td>
<td>12</td>
</tr>
</tbody>
</table>

4.2.3.3 Experimental Results

The self-written Malware is used to test the performance of the model. Above is the table 4.1 showing the result of the testing. The detection rate with self-written malware is 100%.

Also the test has been conducted with real malware in two parts. One is on Steamy Window application with PJApps malware while other is on Monkey Jump 2 application with HongTouTou Malware. The result of this testing is shown in the table 4.2.

4.2.4 Permission Based Android Malware Detection

4.2.4.1 Introduction

There are three different types of malware detection techniques:

- Attack or Invasion detection
  Detect unauthorized access by outsiders.

- Misuse detection (signature-based)
  Detect misuse by insiders. It gives very good detection results for well-known attacks.
• Anomaly detection (behavior-based)

Detects patterns in a given dataset which diverts from normal behavior. Abnormal behavior of the system is estimated by this technique. If a deviation between an instance and normal behavior exceeds a predefined threshold, it generate anomaly alarm

Advantage of Misuse detection

• It has no false positives and can quickly detect intrusion.

Disadvantage of Misuse Detection

• New unfamiliar intrusions cannot be detected. Even little variant of known attacks cannot be detected.

Advantage of Anomaly detection

• Previously unseen intrusion events can be detected potentially

Disadvantage of Anomaly detection

• It require large set of training data to construct the profile showing normal behavior. It also gives many false-positives.

For removing these shortcomings of misuse detection and anomaly detection, profiles should be updated with large amount of the datasets at regular interval of time. A large amount of the datasets also increases the problem of inconsistency, redundancy and ambiguity.

K-Mean Clustering is an unsupervised data mining technique for intrusion detection and it is easy to implement. The drawback of K-Mean are

• Class dominance problem

• Force assignment problem

• No class Problem

The Model proposed by the paper provides solution for removing above drawbacks.
4.2.4.2 How it Works

Model [12] is based on feature selection as a first phase, K-Mean clustering model generation as a second phase, classification of this new dataset which is generated by second phase as third phase.

![Android Malware Detection Process](image)

Figure 4.17: Android Malware Detection Process

4.2.4.3 Features

The model retrieved several selected features from the corresponding application package (APK) file. Example of features can be Permissions to access some specific service. Some features are given as follows:

- android.permission.INTERNET: The application request for Internet using this permission.
- android.permission.CHANGE_CONFIGURATION: The application can request to change configuration files of the mobile devices.

4.2.4.4 Features Extraction

It is done in number of steps as follows:

- The model downloaded and collected malware and goodware applications from application market.
• It decompress applications to extract the content.
• It extract the permission request features from each application.
• It build a dataset in an ARFF file format with the extracted data.

4.2.4.5 Feature Selection Method

Information gain: This method depends on entropy of the attributes and it selects the largest value of gain as the best feature. Gain of an attribute $A$ on a collection of examples $S$ is given by

$$\text{Gain} (S, A) = \text{Entropy} (S) - \sum_{V \in \text{Values}(A)} |S_V| \times \text{Entropy} (S_V)$$

Selected features are divided into Training Data and Test Data by standard machine learning language.

First stage: $k$ disjoint clusters are obtained by performing clustering on training instances. A region of similar instances constitutes each cluster. The instances are similar in terms of Euclidean distances between the instances and their cluster centroids.

Second stage: K-means method is cascaded with decision tree learning by using the instances in each K-means cluster.

4.2.4.6 Experimental Results

The model extracted the necessary features to analyze from sample applications (goodware and malware). The model built dataset in (.arff) file format from the extracted features. The model used these two datasets to distinguish malware and goodware applications by machine learning approaches.

Tables below shows the experimental results.


### Table 4.3: Datasets for Malware Detection Framework

<table>
<thead>
<tr>
<th>Dataset Name</th>
<th>Number of Samples</th>
<th>Number of Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset 1</td>
<td>200</td>
<td>160</td>
</tr>
<tr>
<td>Dataset 2</td>
<td>500</td>
<td>160</td>
</tr>
</tbody>
</table>

### Table 4.4: Experimental Results of Two Datasets

<table>
<thead>
<tr>
<th>Dataset Name</th>
<th>Method Name</th>
<th>TP Rate</th>
<th>FP Rate</th>
<th>Precision</th>
<th>Recall</th>
<th>ROC Area</th>
<th>Correctly Classified Instances(%)</th>
<th>Incorrectly Classified Instances(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset 1</td>
<td>J48</td>
<td>0.967</td>
<td>0.096</td>
<td>0.916</td>
<td>0.907</td>
<td>0.918</td>
<td>90.72%</td>
<td>9.28%</td>
</tr>
<tr>
<td>Dataset 1</td>
<td>Random Forest</td>
<td>0.918</td>
<td>0.081</td>
<td>0.918</td>
<td>0.918</td>
<td>0.954</td>
<td>91.75%</td>
<td>8.25%</td>
</tr>
<tr>
<td>Dataset 1</td>
<td>CART</td>
<td>0.978</td>
<td>0.157</td>
<td>0.849</td>
<td>0.978</td>
<td>0.87</td>
<td>90.72%</td>
<td>9.27%</td>
</tr>
<tr>
<td>Dataset 2</td>
<td>J48</td>
<td>0.88</td>
<td>0.121</td>
<td>0.88</td>
<td>0.88</td>
<td>0.915</td>
<td>88%</td>
<td>12%</td>
</tr>
<tr>
<td>Dataset 2</td>
<td>Random Forest</td>
<td>0.916</td>
<td>0.084</td>
<td>0.916</td>
<td>0.916</td>
<td>0.969</td>
<td>91.58%</td>
<td>8.42%</td>
</tr>
<tr>
<td>Dataset 2</td>
<td>CART</td>
<td>0.851</td>
<td>0.151</td>
<td>0.851</td>
<td>0.851</td>
<td>0.878</td>
<td>85.05%</td>
<td>14.94%</td>
</tr>
</tbody>
</table>

### 4.2.5 Enhancing Security of Linux-based Android Devices

In this paper [11], attention is given on common Linux-tools which can be used as security measure for Android. Besides this, one model based on events occurring in the system, is proposed for Intrusion Detection for Android.

We studied the paper and found some tools of Linux which can be used as a security measures for Android.

- **Anti-Virus tool**
  - Clam AntiVirus: It can be used for email scanning on mail gateway. It works on Android.

- **Firewall**
  - Netfilter: It does not work on Android. The problem is that kernel needs to be recompiled from source. Page alignment issues causes problems in the compilation of “iptables”.

- **Rootkit Detectors**
  - Chkrootkit: It scans for worms, rootkits and Linux Kernel Module (LKM) trojans. It works on Android with minor dependencies.
• Intrusion detection

  – Snort: It excels at traffic analysis and packet logging on IP networks. Static libc requirements causes problems in static compilation. Dependencies to libpcap, libdnet, libnet, pcre and iptables are some of the problems. Besides this it requires statically compiled libc parts which are not available on Android.

• Other Useful Tools

  – Busybox, Bash, strace, OpenSSH and Nmap are some other tools which can be used on Android very easily for various different purposes.

4.2.5.1 Self-built Intrusion Detection System

Figure 4.18: Intrusion Detection System
Figure shows three different levels as follows:

- **Linux Operating System Level**
  - This level provides all the initiated events by Kernel and file system.

- **Linux Application Level**
  It consists of two programs;
  - Monitoring application : Information(features) from the Linux kernel and file system is extracted by this program. The features includes hardware and software state of the device.
  - A control daemon : Status checking and persistence of the monitoring application are the duties of this unit.

- **Parts of Linux Application Level**
  - Interconnect Daemon : Event detection module triggers this deamon for generating vectors containing features
  - Event Detection Module (EDM) : Changes in the kernel and file system are recognized by this module. Corresponding events are generated by this module, e.g. a new process is started. Features are extracted of different size and content, depending on these events.
  - Kernel Monitoring Module : Process lists, system call traces, symbol tables and other kernel-based features are extracted by this module.
  - Filesystem Monitoring Module : File information e.g. a list of open files or an integrity check on predefined files is extracted by this module
  - Log-file Monitoring Module : This module extracts information on changes and existence of the logs
  - Network Monitoring Module : Information related to current network configurations, configuration changes, network status, and network traffic are extracted by this module
  - Database Interface (DBI) : Access to the Android SQLite database from Linux application level is provided by this interface. The feature vectors created by the event detection module are stored here
• Java Application Level

Several tasks for anomaly detection, detection collaboration, and detection response are been realized be the monitoring and detection architecture on Java application layer

– Detection Module: Light-weight detection algorithms based on feature vector excerpts are run by this module. It consist of Detection manager and Detection Plug-ins. These plugins are used for analyzing feature vectors and results from different detection algorithms

– Collaboration Module: It enable detection in a collaborative manner as an API

– Response Module: Countermeasures to detected incidents are enabled by this module

– Communication Module: This module provides suitable functions and network access for exchanging feature vectors with the remote server or collaborative peers

– Java Database Interface (JDBI): Access to the Android SQLite database from Java application level is provided by this module. It extracts feature vector and detection result recorded by the system

– Graphical User Interface: It visualizes current monitoring, detection, collaboration, and response status

Detecting Anomalies

Two principal techniques for protection against unwanted software and intrusion are present; misuse detection and anomaly detection.

• Misuse Detection: Known signatures of malware and attacks are intended to recognize by this method.

• Anomaly detection: Determining the degree of normality of some observables is the role of this method.

The model is based on anomaly detection. However detection architecture is not required to be changed for misuse detection

As mobile phones are having too much constraints in terms of battery memory etc. The complex computation tasks and huge data is oursourced on an external remote server. The model is an event-based
Detection Mechanism

Following tasks are handled for detecting the intrusion

- Event perception carried out by event sensor (event detection module (EDM))
- To gain informations about some system observables (features), System monitoring is preformed, when required.
- Detection, i.e. assigning a level of normality by properly analyzing system features. The module consists of a detection manager, detection and meta detection units which are event specific.
- Learning, which the external server is responsible for.
- In the absence of external server Collaboration is needed. Sometimes it is used in the presence of external server to reduce load
The highest priority is given to the daemon, detection manager. A method `setPersistent()` prevents even the system from stopping it.

Every detection module sends signals to the Detection manager. It then starts the proper detection unit to deal with it, which are implemented as sub-activities. They assign to each feature vector a level of abnormality and then return them. An alert is given out by the detection manager to the user if a predefined specific threshold is crossed.

All the statistical learning is done on the server. The server does the statistical learning from the training data from the database of the mobile server. It then sends to the mobile device the updated parameters.

For example, a new process launches if one of the described events occurs. The event detection module senses the event and the system monitor and the detection manager are informed about

Figure 4.19: Intrusion Detection System Architecture
the kind of event. Some system features related to the event is extracted by the system monitor, such as the CPU/memory utilization, process data and the sequence of system calls. As per the design, an event detection unit corresponding to the process started event is launched by the detection manager. This unit takes the feature vector from the system monitor and then evaluates the level of alert.
Chapter 5

Comparison

The comparison shows that TISSA is a much more efficient model than Dr. Android and Mr. Hide. One more substantial difference is that it does not modify the apk of application while the other model is modifying it. The advantage of Dr. Android and Mr. Hide over TISSA is that, it allows the user to lower the privilege level of permission, e.g. it restricts a particular application to access the internet with lower privilege, having access of only domain d and not of the full internet. While in TISSA, full internet permissions need to be blocked.

Mockdroid can be successfully integrated in TISSA. The difference between Mockdroid and TISSA is that the permission which are mocked in Mockdroid can successfully be rolled back and legitimate permissions can be given while using the application which can not be done in TISSA.

Table 5.1: Breakdown of Android Modification in TISSA

<table>
<thead>
<tr>
<th>Component Name</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Privacy Setting Provider</td>
<td>330</td>
</tr>
<tr>
<td>Application Management Program</td>
<td>452</td>
</tr>
<tr>
<td>LocationManager and LocationManagerService</td>
<td>97</td>
</tr>
<tr>
<td>TelephonyManager</td>
<td>61</td>
</tr>
<tr>
<td>ContactsProvider (Contacts and Call Log)</td>
<td>48</td>
</tr>
<tr>
<td>Total</td>
<td>988</td>
</tr>
</tbody>
</table>
Table 5.2: Microbenchmark performance results of Dr.Android and Mr. Hide

<table>
<thead>
<tr>
<th>Task</th>
<th>Orig (s)</th>
<th>Transformed (s)</th>
<th>Slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet</td>
<td>16.241</td>
<td>20.252</td>
<td>25%</td>
</tr>
<tr>
<td>Location</td>
<td>15.004</td>
<td>19.407</td>
<td>29%</td>
</tr>
<tr>
<td>Ringtone</td>
<td>1.257</td>
<td>1.382</td>
<td>10%</td>
</tr>
<tr>
<td>Contacts</td>
<td>0.634</td>
<td>0.953</td>
<td>50%</td>
</tr>
</tbody>
</table>

The concepts of Formal Verification and SORBET can be used to find which permissions are used frequently by most of the applications so that those permissions can be modified first with the model suggested by TISSA along with the integration of Mockdroid.

We studied different papers as explained in section 4.2 for attack / intrusion detection. In all papers, the common thing was the constraints of mobile devices and smartphones because of which attack / intrusion detection cannot be carried out appropriately on them.

Hence, it is the good idea to carry all such mechanisms on a remote server in order to improve efficiency of devices in terms of power memory and performance. We also tested the device Samsung Galaxy Y for different detection mechanisms as shown in the Table 3.1.

It can be seen that such mechanisms reduce the performance of devices heavily and hence it is always efficient to perform such mechanisms on remote servers.

The attack detection mechanism like antivirus which are tested and shown in table 3.1 are Signature-based. Signature-based detection works by scanning the contents of computer files and cross-referencing their contents with the code signatures belonging to known viruses. A library of known code signatures is updated and refreshed constantly by the anti-virus software vendor.

Hence, they are not able to detect attacks, viruses, trojans which are very new and whose signatures [17] are not available. Research has been done and shown that signature for virus can be made available in 48 hours of release of particular virus. So, the device will be vulnerable to attacks till 48 hours.
Zero-day attacks are also possible while using such attack detection techniques. It is an attack that exploits a previously unknown vulnerability in a computer application, meaning that the attack occurs on “day zero” of the awareness of the vulnerability.

As we explained in section 4.2, in various papers the attack is being detected using machine learning techniques i.e Behavior-based or event-based approach. Such techniques can be used to detect above mentioned and similar attacks. Though these techniques also have some disadvantages but these disadvantages are overshadowed by the advantages of such mechanisms while detecting the attacks.
Chapter 6

Problem Statement

Our aim is to design and implement a model suggested by Tissa for Android users, specially for Aakash Tablet. We will also integrate the model suggested by Mockdroid in our implementation of TISSA model, so that the user data can be secure efficiently and some real run-time control can be given to the user at the same time.

TISSA is just a prototype. It is still not available for Android users. We will implement the suggested model and release it as Open Source for Android users. Though, Mockdroid is available in Open Source, it is available as an independent model. So we will integrate it with our implementation of TISSA model, so that the security of Android user will increase without much changes in the behavior of the applications.
Chapter 7

Implementation

As we proposed to implement TISSA model along with the integration of Mockdroid to provide the run-time control to the user, we implemented the same model for the following permissions [7]:

- GET_TASKS
- GET_ACCOUNTS
- READ_SMS
- READ_PROFILE
- CHANGE_CONFIGURATION
- BATTERY_STATS

Following are the details about each Permission.

- GET_TASKS
  It allows an application to get information about the currently or recently running tasks.

- GET_ACCOUNTS
  It allows access to the list of accounts in the Accounts Service.

- READ_SMS
  It allows an application to read SMS messages.

- READ_PROFILE
  It allows an application to read the user’s personal profile data.
• CHANGE\_CONFIGURATION

Allows an application to modify the current configuration, such as locale.

• GET\_ACCOUNTS

Allows access to the list of accounts in the Accounts Service

Any application can ask the above mentioned permission at the time of installation. Once the user allows the access, the application can misuse the permission granted to steal the data or in other sense. We have given user, the control to deny the access to the permission even while installing and running the application.

We changed the coarse-grained functionality of Android Permission Model and provide some fine-grain mechanism using TISSA prototype.

We also provide the run-time control of Mockdroid Model. What we mean by run-time control is explained by an example as follows.

Lets say Person ‘A’ is playing a game on internet. While installing the game application, the access to internet was asked, which the user did not give. The user scores highest in the game and wants to post the score online for which internet is required. So, at the run-time the application will get paused and user will be able to switch the internet permission to legitimate and can post the score. Later the user can switch to the previous secured mode.

Though we have not implemented our model for internet permission, the same run-time control is given for other permissions which we implemented.

Following are the screenshots while performing the experiments on our TISSA model for READ\_SMS permission.
CHAPTER 7. IMPLEMENTATION

Figure 7.1: Permission set to ‘Trusted’

Figure 7.2: Displaying Original Message
Figure 7.3: permission set to ‘Bogus’

Figure 7.4: Displaying Bogus Message
Chapter 8

Future Work and Conclusion

We studied various papers and research work, and find that the permission model on which the current Android system is working, is not good enough to protect the private data of the user. The Android system is working on coarse-grained permission model while in order to protect the leakage of users private data, it is really necessary to implement some fine-grained permission mechanism in which the user can have more power rather than the application itself. We compare all the ideas proposed by the paper studied in terms of performance issues, flexibility, and feasibility and finally concluded to implement the model suggested by TISSA along with the integration of Mockdroid. We implemented the TISSA model for six permissions and given the run-time control to change the permission whenever required by the user.

We studied various papers and provide the platform to implement the attack detection mechanism effectively and efficiently. We can further implement such mechanism in which behavior-based approach will be applied to the datasets collected from the device. The whole process can be performed on remote server.

We can made it available for Aakash tablets in IIT Bombay. By assigning one server for replicas of tablets, we can apply different attack detection mechanism.
Bibliography


[8] Jinseong Jeon, Kristopher K. Micinski, Jeffrey A. Vaughan, Ari Fogel, Nikhilesh Reddy, Jeffrey S. Foster, and Todd Millstein. Dr. android and mr. hide: fine-grained permissions in


