 ISSN: 2454-4248
12 – 22

Android Forensics and It's Existing Vulnerabilities Penetration Testing Framework

Praful Meshram
Research Scholar
SGGS Institute of Engineering & Technology, Nanded
praful.meshram@gmail.com

Chetan Janbandhu
Automation Platform Developer
Bell Labs, New Jersey, United States of America
chetan.janbandhu@nokia.com

Abstract – Smartphones are also portable computers as they provide many services needed in our day to day lives such as texts, calls, camera, Bluetooth, GPS and various other applications. Due to the attractive features of android smartphones, it's use is increasing tremendously. With the growing popularity of Android and it being one of the best players in mobile industry, knowing the best practices for its security becomes very crucial. Android is known as a platform that lends itself to hacking. Smartphones are prone to data leakage as they can easily exchange data over the Internet. Applications are made of four components namely Activity, Service, Broadcast receivers, and Content provider. This paper proposes the various threats and security risks Activity, Broadcast Receivers and Content Providers pose and how they can be responsible for sensitive data leakage without the user's knowledge. It also states the various attacks and the prevention mechanism and focuses on the prevention mechanism known as YASSE. This paper also states various other methods and technologies that are implemented for cloud based security that not only enhances the safety of the devices, but also reduces the system load of the devices.

Keywords *- Android, attacks, security, malware, frameworks.*

*****

I. INTRODUCTION

Android is a widely popular and open source operating system designed for smartphones and other mobile devices. While Android is based on Linux, it defines an entirely new middle-ware and GUI environment in which applications execute. Applications are mostly written in Java, which is compiled to Dalvik bytecode, which runs in a virtual machine similar to the Java virtual machine. Apart from Java, Android also allows parts of apps to be coded in native code.

Inter-Process Communication (IPC): In Android, system services are provided in separated processes, with a convenient IPC mechanism (Binder) to facilitate the communication among system services and applications. Binder IPC is heavily used in Android and recommended in the design of applications.

Android employs a quite efficient and convenient IPC mechanism, Binder, which is extensively used for interaction between applications as well as for application-OS interfaces. Binder is implemented as a kernel driver and user-level applications could just interact with it through standard system calls, e.g., open(), ioctl(). Binder is the key infrastructure of Android system and aggressively used to connect various parts of the system together. To facilitate resource accessing from isolated applications and data sharing among applications and the system, Android designs
a permission-based security mechanism. Each application needs permissions to access system resources. These permissions are granted from users at install time. At runtime, each application is checked by Android before accessing sensitive resources. Any access to resources without granted permissions will be denied. The permission mechanism in Android is fine-grained which is different from iOS. In Android 4.2, there are 130 items of sensitive resources that are protected with permissions.

Fig. 1 Android Architecture

The rapid growth of smart phones has lead to a renaissance for mobile services. Smart mobile devices include smart phones, PDAs, tablet PCs and so on. People use them to store personal information, send and receive emails, browse the web,process documents and entertain. According to the Gartner research results, the android system accounts for more than 50% of the global smart mobile devices operating systems. The openness of android and its large market share make android the most vulnerable system. Android phones store a lot of private information of users in its database. Android uses permissions to protect sensitive resources from untrusted apps. However at the time of installation, the user grants permission to these apps to access various contents. After permission is granted at install time, the apps could use these permissions without any further restrictions. These apps keep and manage sensitive data such as address book, photos, etc. Smartphones are prone to data leakage as they can easily exchange data over the Internet. Android relies on the Sandbox to protect data of one app from another app while it leaks sensitive data due to Content Provides and Broadcast Receivers.

 ISSN: 2454-4248
12 – 22

A. Overview

Applications are made of four components namely Activity,Service, Broadcast receivers, and Content provider. The Activity defines the GUI and its interaction with the user. The Service runs in the background performing long run applications. The Broadcast receiver responds to specific system-wide messages and the Content Provider is responsible for sharing or managing the data between applications.

(1*) Activity*:

An Activity defines a graphical user interface and its interactions with the user's actions. A malicious app has the ability to create a complete virtual environment that acts as a full Android interface,with complete control of all user interactions and inputs. This makes it very hard for the victim to escape the grip of the attacker. These attacks are often called the GUI attacks.

There are various attack vectors that a malicious app can use to mount GUI confusion attacks. Other enhancing techniques have also been identified that do not present a GUI security risk in themselves,but can assist in making attacks more stealthier.

However, before discussing about the various attacks, the typical android user interface appearance needs to be identified.

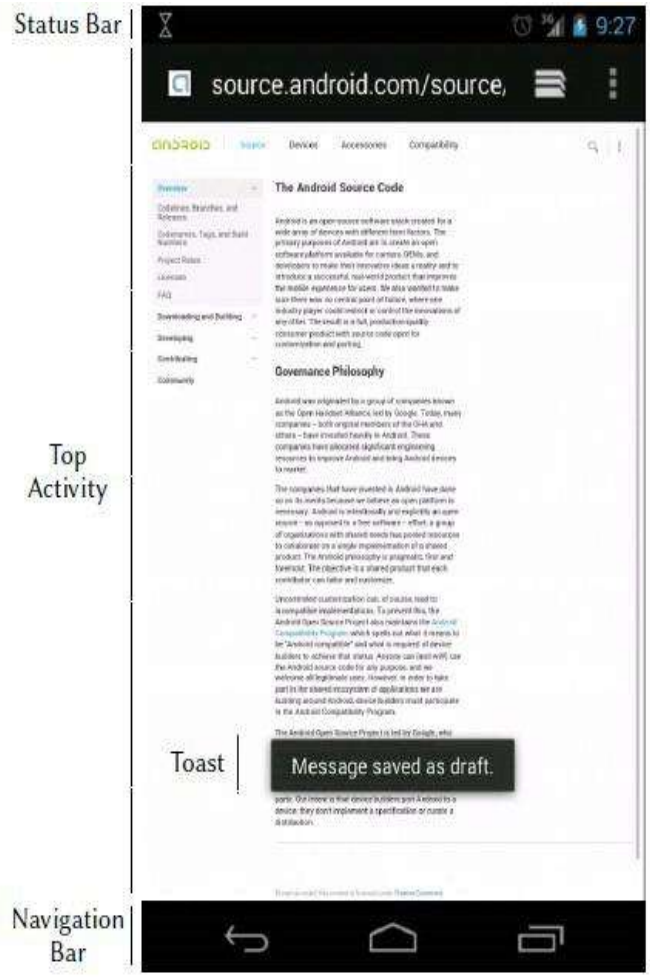

Fig. 2 Overview of Screen

Status Bar-shows the information about the device's state such as current network connectivity status or the battery level.

Navigation Bar-allows users to navigate among current apps as well as within the focused app. The Back button removes top Activity from the top of the stack. The Home button lets the user return to the base screen. The Recent button shows the list of top Activities of the running apps,so the user can switch among them.

Apps can draw graphical elements with the following components - Views, Activities, Windows.

- Views - A View is the basic UI building block in Android. Buttons, text-fields, images,and OpenGL view ports are all examples of views.
- Activities - An Activity can be described as a controller in a Model-View-Controller pattern. An Activity is usually associated with a View(for the graphical layout)and defines actions that happen when the View elements are activated(e.g.,a button gets clicked). Activities are organized in a global stack that is managed by the Activity Manager system Service. The Activity on top of the stack is shown to the user.
- Windows - A Window is a lower-level concept:a virtual surface where graphical content is drawn as defined by the contained Views. Windows are normally managed automatically by the WindowManager system Service, although apps can also explicitly createWindows.

(1.1) *GUI Confusion Attacks-*

- Draw on top : Attacks in this category aim to draw graphical elements over other apps. Typically,this is done by adding graphical elements in a Window placed over the top Activity. The Activity itself is not replaced, but malware can cover it either completely or partially and change the interpretation the user will give to certain elements. Apps can explicitly open new Windows and draw content in them using the addView API exposed by the WindowManager Service.
- App switch : Attacks that belong to this category aim to steal focus from the top app. This is achieved when the malicious app seizes the top Activity:that is,the malicious app replaces the legitimate topActivity with one of its own.

(1.2) *Enhancing techniques-*

Techniques to detect how the user is currently interacting with the system: To use the described attack vectors more effectively,it is useful for an attacker to know how the user is currently interacting with the device.

Reading the system log-Android implements a system log where standard apps,as well as system Services,write logging and debugging information. This log is readable by any app having the relatively-common READ LOGS permission. By reading messages written by the ActivityManager Service,an app can learn about the last Activity that has been drawn on the screen.

Moreover, apps can write arbitrary messages into the system log and this is a common channel used by developers to receive debug information. We have observed that this message logging is very commonly left enabled even when apps are released to the public, and this may help attackers time their actions,better reproduce the status of an app, or even directly gather sensitive information if debug messages contain confidential data items.

(2) *Service*:
Services run in the background and do not interact with the user. Downloading a file or decompressing an archive are examples of operations that may take place in a Service. Other components can bind to a Service, which lets the binder invoke methods that aredeclared in the target Service's interface. Intents are used to start and bind to Services. Services run with a higher priority than inactive or invisible activities and therefore it is less likely that the Android system terminates them for resource management. The only reason Android will stop a Service prematurely is to provide additional resources for a foreground component usually an Activity. When this happens, your Service can be configured to restart automatically.

(3) *Broadcast Receiver*:
The Broadcast receivers are used as a mean of communication between app's components, between different apps, and between the OS and apps. This Android component enables applications to register for system or application events or actions (e.g. receive call, receive message).Once an event occurs, Android runtime notifies all applications that have a registered receivers of that particular action. A broadcast is a message that any application can register to receive. The Android system delivers numerous broadcasts for system events, such as when an Internet connection is enabled or a new SMS arrives. While Broadcast receivers are considered a useful feature by developers, users' experience and privacy might be affected negatively by them. They provide hackers with the ability to orchestrate their attacks and link them with the events that are happening on the mobile device.
With the use of Broadcast receivers the location privacy of users can be compromised, moreover,the usage of Broadcast receivers by malicious applications is remarkably higher than benign applications.

(4) *Content Providers*:
Android relies on the Sandbox to protect data of one app from another app while it offers the ContentProviders to share databases. A ContentProvider manages access to a central repository of data. It can be used to manage access to a variety of data storage sources. It is the best way to share data among applications.Content Providers are databases addressable by their application-defined URIs. ContentProviders are based on the client- server model. Server apps assign URIs to identify their databases externally. Client apps are supposed to know the names, table structures and URI of the database to use. Client apps use ContentResolvers to communicate with server side ContentProvider. The Content Resolver accepts requests from clients, and resolves these requests by directing them to the content provider with a distinct authority. To do this, the Content Resolver stores a mapping from authorities to Content Providers. Then the ContentProvider receive query results from database via DBHelper and send them back to ContentResolvers.In other words, ContentProvider and ContentResolvers serve as windows for sharing databases between server and client apps.

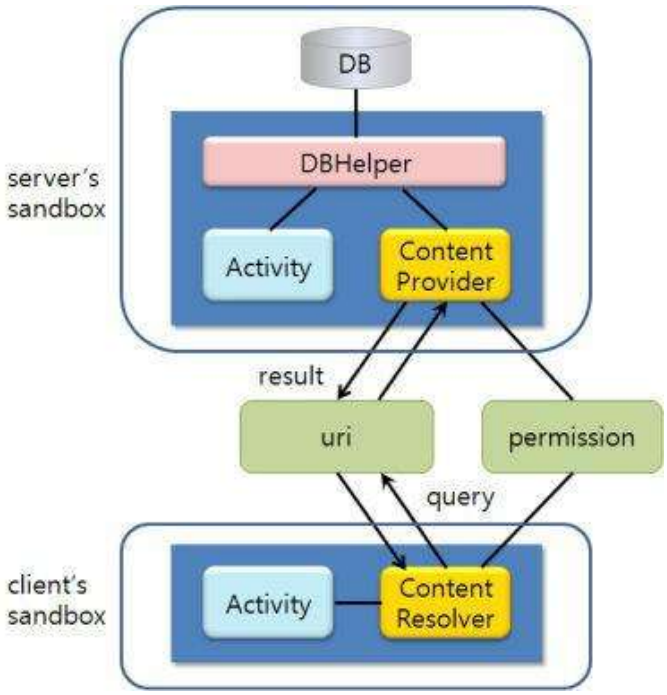

Fig 3:Database sharing between applications.

To use major system resources or functions, Android apps should make requests for permissions of Android system via AndroidManifest.xml. For instance, to capture images from the camera on a Smartphone, an app should get the android.permission-group.CAMERA permission from the Android system. Likewise, to use the Internet,an app should get the android.permission.INTERNET permission from the Android system. Android defines and offers over 200 permissions and enables developers to define and use their own permissions. Server apps can use permissions to restrict accesses to their databases. In general, server apps define in AndroidManifest.xml their specific permissions for reading and writing to share their databases. Client apps describe in AndroidManifest.xml the permissions assigned to databases they intend to use and make requests of the system. Once client apps run and ContentResolvers send queries to ContentProvider, Android system checks out if client apps own permissions required to access such databases. Unless client apps own permissions required to access those databases, the service is denied.

B. Intents

An Intent is a messaging object you can use to request an action from another app component.

There are two types of intents:

- Explicit intents specify the component to start by name (the fully-qualified class name). You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start. For example, you can start a new activity in response to a user action or start a service to download a file in the background.
- Implicit intents do not name a specific component, but instead declare a general action to perform, which allows a

 ISSN: 2454-4248
12 – 22

component from another app to handle it. For example, if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.

Figure shows how an intent is delivered to start an activity.

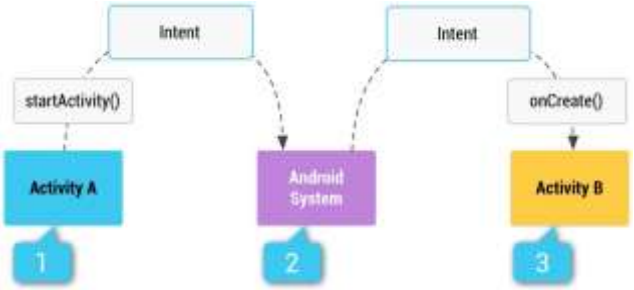

Fig 4:Intents

In Fig4. How an implicit intent is delivered through the system to start another activity: [1] Activity A creates an

Intent with an action description and passes it to startActivity(). [2] The Android System searches all apps for an intent filter that matches the intent. When a match is found, [3] the system starts the matching activity (Activity B) by invoking its onCreate() method and passing it the Intent.

II. ANDROID SECURITY MODEL

Android is a widely popular and open source operating system designed for smartphones and other mobile devices. While Android is based on Linux, it defines an entirely new middle-ware and GUI environment in which applications execute. Applications are mostly written in Java, which is compiled to Dalvik bytecode, which runs in a virtual machine similar to the Java virtual machine. Apart from Java, Android also allows parts of apps to be coded in native code.

Inter -Process Communication (IPC): In Android, system services are provided in separated processes, with a convenient IPC mechanism (Binder) to facilitate the communication among system services and applications. Binder IPC is heavily used in Android and recommended in the design of applications.

III. ANDROID ATTACKS AND MALWARE

A. Attacks

Android is open source software stack based on Linux kernel for its services. The source code is available for each of the android functionality. So attackers can find flaws in the operating system to attack the device. There are various techniques that can be used to attack the android smart phone. There are three methods which are used mostly that are Drive by download, Update attack and application repackaging. Most of the applications on the android market (both official and other) are repackaged. In repackaging the attacker have to download the application from android market, perform reverse engineering ,insert the malicious code into the application and upload it to market. The Users who download this kind of repackaged application compromise their privacy and can lose the private data or the confidential information. Whereas in Update attack the application seems legitimate at the time of installation. After the installation application asks for update. As the user updates the application the malicious payload get downloaded in the application on the device. The Drive by download method works differently such as user is playing the game and he got the ads posing that to unlock some attractive features of game click here. As user click theadvertise the malware get downloaded on the device. Attacker can leverage user to malicious sites by posting links within SMS or Email.

In this section we will describe the reverse engineering, the insertion of malicious code into Android banking Application and DDOS attack.

(1) Reverse Engineering

Reverse engineering is the acquisition of a deep understanding of processes of a system or application by having only the finished product. Being able to understand the functioning of an application you do not have the source code is the base Reverse Engineering. Reverse engineering

that we will perform in this paper concerns the android banking applications. So we must first of all have the APK (Android Application Package) of application. This can be retrieved from a device or from the Play Store. The apk file is a file that contains all the files needed to run an android application such as :

Dalvik executable (.dex) file : is the executable file that results from the compilation of java source code Manifest file : is a file that contains the setup required by android applications Resources file: is a file that contains the resource

required to run the application as images. The reverse engineering allows us to have access to all the files of applications that serves us in our attack. After downloading android target applications, we will proceed to the reverse engineering. The purpose of reverse engineering is to convert downloaded files.apk into files.java to access the source code. First we used the tool Apktool to performe reverse engineering on .apk file, so we got the byte code of the application file.class. Then we used the JD-UI tool to visualize and convert file.class to file.java. This procedure allowed us to obtain the source code of the banking.

Insert malicious code into Android banking app The second phase of the attack is to infect the source code, and add malicious activity. In order to deploy the infected applications, it is necessary to sign them to deceive mobile end-users. For this we used keytool to generate the private key and jarsigner to sign the modified application. After finishing the changes on the source code, we converted those infected files to .apk files and deployed them. In a concern for ethics and in order to not infect the existing network we deployed these applications on a controlled local network in our laboratory. After having downloaded the applications, end users will be automatically infected. So we can perform several types of attacks based on the type of malicious code inserted. In our case we insert malicious activity that can overload the network and launch a DDOS attack on remote servers including banking servers, which can cause serious damage to the target bank and all its customers.

(2) DDos Attack

A DDOS attack is a DOS form attack that is not performed by a single node but achieved by the combination of multiple nodes. All nodes simultaneously attack the victim node or target network by sending them huge amount of packets, this will totally consume the bandwidth of the victim and this will not allow the victim to receive legitimate traffic. So our DDOS attack is performed as follow; After installing our forged application on the device, our malicious activity establishes a connection with the server controlled by the attacker by sending an echo request and the server responds with an echo reply. Then we launch TCP flood attack. A flood attack involves the zombies sending large volumes of traffic

to a banking server. We generate several TCP SYN flood attack with a maximum connection rate of 500kbps on the server of the bank in order to saturate the bandwidth of the bank server.

B. Proposed solution

In this section we propose some solutions to protect mobile applications and banking server against such types of attacks

(1) Securing banking application

As we have seen in previous sections, compiled Android applications could be decompiled to obtain the source code of the application. So to make the understanding and deep analysis of the code more difficult, the obfuscation techniques could be used. There are several tools that help obfuscate the source code of an application. The default tool proposed by android is proguard tool that shrinks, optimizes, and obfuscates your code by removing unused code and renaming classes, fields, and methods with semantically obscure names. The result is a smaller sized .apk file that is more difficult to find by reverse engineering. The paid version of this tool is DexGuard which provides advanced security features tailor-made for the Android operating system. It offers protection against static analysis. DexGuard shields your apps from cloning, tampering, key extraction and piracy, by applying multiple techniques such as string encryption, class encryption, asset encryption, call hiding, code obfuscation and resource obfuscation

(2) Securing banking server

Secure banking server against DDOS attack: To secure banking applications servers against these type of attacks we propose the installation of an intrusion detection system (IDS) this will help to detect and mitigate the illicit network activities. This detection system often keeps files

of system activity statistics, user connections, disk activity, etc. In addition, alarms are triggered when abnormal activity occurs. Most intrusion detection systems also try to match network traffic with fingerprints of known attacks. Moreover, some "alarm" are triggered when a potential attack

is detected, then the system records the event, notifies the system administrator via email or pager. It is important to know that an intrusion detection system can operate in either a final machine or a dedicated machine on the network.

1) solution Authentication: Authentication aims to verify the identity of an entity so as to allow this entity to resources (system, network or server). So, we propose strong authentication for mobile applications on banks of servers in order to identify the credibility of two different entities.

This mechanism will ensure the traceability, integrity, confidentiality and authorization in which we guarantee the security of both sides (client / server). In order to secure applications against such attacks it is necessary to proceed on several axes. First we must protect the source code against reverse engineering and secondly we must protect appli-cations from cloning, tampering, key extraction and piracy. We used in our tests the DexGuard tool to realize obfuscation and so making difficult understanding and analyzing the code. We must also think about securing the network against this type of infection. For that we must establish strong authentication mechanisms between the server and end users, encrypt communications and deploy IDS and firewalls throughout the network.

C. Malware

Software that "deliberately fulfills the harmful intent of an attacker" is referred to as malicious software or malware. These are intended to gain access to device systems and network resources, disturb device operations, and gather personal information without taking the consent of the user, thus creating a menace to the availability of the Internet, integrity of its hosts, and the privacy of its users. Malwares come in wide range of variations like Virus, Worm, Trojan-horse, Rootkit, Backdoor, Botnet, Spyware, Adware etc. These classes of malwares are not mutually exclusive meaning thereby that a particular malware may reveal the characteristics of multiple classes at the same time.

Attackers exploit vulnerabilities in web services, browsers and operating systems, or use social engineering techniques to make users run the malicious code in order to spread malwares. Malware authors use obfuscation techniques like dead code insertion, register reassignment, subroutine reordering, instruction substitution, code transposition, and code integration to evade detection by traditional defenses like firewalls, Antivirus and gateways which typically use signature based techniques and are unable to detect the previously unseen malicious executables. To overcome the limitation of signature based methods, malware analysis techniques are being followed, which can be either static or dynamic. The malicious program and its capabilities can be observed either by examining its code or by executing it in a safe environment.

Android is a widely popular and open source operating system designed for smartphones and other mobile devices. While Android is based on Linux, it defines an entirely new middle-ware and GUI environment in which applications execute. Applications are mostly written in Java, which is compiled to Dalvik bytecode, which runs in a virtual machine similar to the Java virtual machine. Apart from Java, Android also allows parts of apps to be coded in native code.

(1*) Static Analysis:*

Analyzing malicious software without executing it is called static analysis. The detection patterns used in static analysis include string signature, byte-sequence n-grams, syntactic library call, control flow graph and opcode (operational code) frequency distribution etc. William Enck proposed a static analysis method to analyze the permissions of an Android application. They first analyze the permissions granted to the application when it is running. Then they compare them with the permissions declared in the AndroidManifest.xml. If they are different, the application maybe hides dangerous permissions. So the installation is prohibited. But this method can only analyze if there are hidden permissions in an application. If there are no hidden permissions, the user determines whether to continue installing the application all by himself. Even though an application declares a number of extremely dangerous permissions (e.g. CALL_PHONE, SEND_SMS), if the application does not hide these permissions, the detection method does not prohibit the installation of the application. While the vast majority of Android users don't know that these permissions may cause serious consequences and continue installing these malicious applications.

 ISSN: 2454-4248
12 – 22

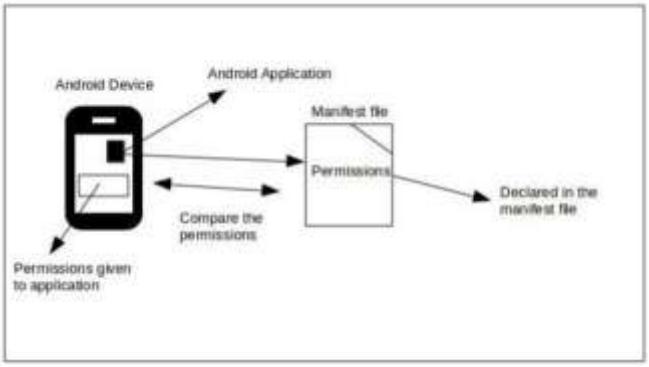

Fig. 5 Static Analysis Proposed By William Enck

(2*) Dynamic Analysis:*

Analyzing the behavior of a malicious code (interaction with the system) while it is being executed in a controlled environment (virtual machine, simulator, emulator, sandbox etc.) is called dynamic analysis.

Thomas Blasing proposed a method to detect malicious behavior by using emulator. They use a monkey which can generate pseudo-random streams of user events such as clicks, touches, or gestures, as well as a number of system-level events. Then they place a sandbox in kernel space to log the behavior of the application at the system level. They analyze the resulting log file to determine if the application is malicious. The method is for security vendors to detect malicious behavior of new applications. But it is not adapted to ordinary mobile devices.

IV. PROPOSED FRAMEWORKS

A. YAASE

Android Security model mostly relies on application

sandboxing , but there have been believable reports that this makes it vulnerable to privilege spreading attacks. Privilege spreading attack: An attack of this type , an application that is under privileges exploits the permissions or privileges of a privileged application. If a malicious application is leveraging the vulnerability of a legit application it is often referred to as confused deputy attack.

Yet Another Security Extension is a fine grained security framework which is transparent to applications where policies define how resources of the phone will be accessed. Yaase uses user-defined label for data to utmost use and enforces the right distribution of tagged data for 2 types of communication -

- Application-to-application
- Application-to-internet

In this framework another a major role is played by a system - Taintdroid , leveraging on its tainting capability and extending it to support user defined labels and for performing modifications due to enforcement of policies on the tagged data.

Yaase policies define the data labels that an application is authorized to have access to by the user. It means , that if an application tries to access a data tagged with label for which that application has no authorization , the application will not be granted access to it.

(1) Architecture of YAASE:

In order to make a security framework so fine grained and transparent to applications some modifications were made in the android components as stated by the authors of YAASE. The modified android components are marked in gray while blue marked are components introduced by YAASE.

To track the data stored on the phone that is accessed by the applications and to see how this data is disseminated both within the phone (i.e., from one application to another)and when the data leaves the device (i.e., an application sends the data over the Internet).The capabilities of TaintDroid have been extended to be able to tag sensitive information with a taint value that is defined in our system component - Labeling store. Every Taint is 32 bit value and is supposed to define following - Control group , Taint Label , and some extra information for history inspection. The control group is supposed to specify the whether the data in question is coming from a system source like GPS.

Once the data is tainted , it has to disseminate according to the users requirements . User requirements are set by the policy provider that sets policies per application. To be able to successfully enforce the policies defined by the Policy provider and take appropriate action on the data , system has hooks called Policy Enforcement Point in the lib binder module this helps in controlling access to simple resources like IMEI ,location data etc.

In the LibBinder, we intercept the standard cursor from where we extract the CursorWindow. The CursorWindow provides methods that can be used for modifying the data contained in the cursor. Using the CursorWindow allows us to filter out from the cursor data only part of the information.

In this way, our enforcement mechanism achieves a fine-grained filter capability. The actions that can be invoked by the policies are defined in the Action Library. Another PEP is placed in the Java Framework Library (JFL) of the Dalvik Virtual Machine for capturing operations on the file system (such as reading and writing on the local storage as well as accessing the phone camera and microphone) and on the network stack for controlling network traffic even if sent over an encrypted socket (SSL). We have modified the socket.open(address) method to inspect the address to where the data is sent. In this way, we can restrict the use of only authorized addresses or substitute the address specified by the application with an address defined by the user. By modifying the sendStream() we are able to intercept the data before it is sent and perform some actions, such as filtering or substitutions.

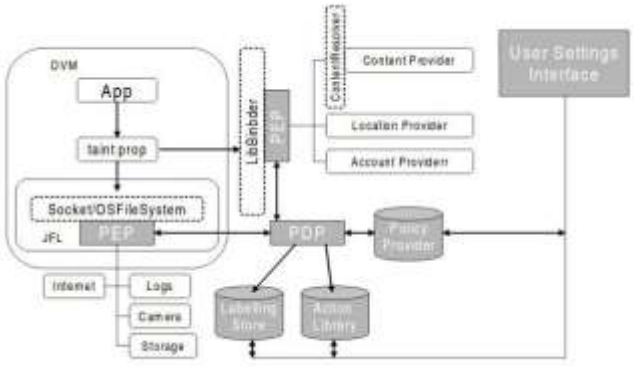

Fig. 6 The YAASE architecture

The role of PEP and PDP goes hand in hand. When an applications needs an access to any of the resources , the PEP checks the request , collects information about the application UID , the resource being requested and type of operation requester wants to do and the tag (if available) with the data. On collecting this

 ISSN: 2454-4248
12 – 22

information PEP forwards it to the PDP - Policy decision point. PDP is responsible for making decisions about whether to allow the request or to perform some action on the data before returning it. PDP uses information collected by the PEP to get the exact policies associated to this application from the policy provider. On evaluating policies retrieved from the policy provider the further decision by PDP is made.

(2) *Policy Generation:*

A brief discussion on policy language will make it clear how a policy is made for controlling data dissemination. Policies are identified by a name and define what operations a requester application can do on a resource. In YAASE a resource can be either a content and service providers in the phone or other applications that expose services for other applications to be invoked. A policy has a **have to perform** that specifies action that have to be performed if it is enforced. There are some libraries present that are equipped to perform several actions on the data and changing the values of parameters of operation being executed. **handle** clause defines an expression on labels associated with data passing from requester to resource if operation is a setter method which does not return any data. There is also a getter method that provides data from resources to requester . **handle** clause refers to different labels to be handled.

With examples it will be clearer-

```
PolicyName: Requester can do operation on
Resource

    2 [have to perform action]

    3 handle dataLabelExpression
```

Fig. 7 The syntax for policy language

```
1 AppPolicy1: OrgContacts can do getContacts on Contacts
2 have to perform filterOut(''Private'', returnData);
3 handle ''Pub'' and ''Contact''
4
5 AppPolicy2: OrgContacts can do send on Internet
6 have to perform sendOnlyTo(aCloudUrl);
7 handle ''Public'' and ''Contact''
```

Fig. 8 Policies applied for orgContacts

```
1 AppPolicy1: OrgContacts can do getContacts
on Contacts
2 handle ''Contact''
3
4 AppPolicy2: OrgContacts can do send on
Internet
```

Fig. 8.1 Policies applied for org Contacts

```
1 AppPolicy1: OrgContacts can do getContacts on
Contacts
2 handle ''Public'' and '' Contact''
3
4AppPolicy2: OrgContacts can do send on
Internet
5 handle ''Public'' and '' Contact''
```

Fig. 8.2 Policies applied for orgContacts

```
1 AppPolicy1: OrgContact can do getContacts on Contact
2 have to perform filterOut(''Private'', returnData);
3 handle ''Public'' and '' Contact''
4
5AppPolicy2: OrgContacts can do send on Internet
6 have to perform sendOnlyTo(aCloudUrl );
7 handle ''Public'' and '' Contact''
```

Fig. 8.3 Policies applied for orgContacts

Let us consider an Organizer application (OrgContacts) providing a smart way of organising the user's contacts and a back-up capability that stores the user's contacts over a cloud service. For its functionalities, the OrgContacts requires access rights to the phone contact provider and to the internet service. In this scenario, the user wants that only her work contacts are accessible to the OrgContacts. Moreover, the user wants to make sure that the OrgContacts sends the contacts to a specific location over the internet. The user's requirements can be expressed by two policies as in Figure 3.

In particular, the policy AppPolicy1 specifies that the OrgContacts can perform the getContacts operation on the contact provider. We assume the user has tagged each contact entry with a label to indicate whether it is a working contact (associated with label ''Public'') or a private contact (associated with label ''Private'').1 The contacts are returned to the OrgContacts through a cursor containing all the entries. This means that the cursor will be tagged with labels "Public" for the working entries, "Private" for the private entries, and "Contact" to indicate it contains data coming from the contact provider. To remove from the contacts private entries, the policy invokes the filterOut action on the returned cursor (indicated as returnData)removing all the entries tagged with label ''Private''. After the execution of the filterOut action, the new cursor is now tagged with only "Public" and "Contacts" labels. At this point, the condition specified by the handle on line 3 is satisfied and the data can be returned to the OrgContacts.

If the **have to perform** clause had not been specified, then the cursor would have been tagged also with label "Private". Because the expression of the **handle** clause defines which are the only labels permitted then the policy would have denied the returning of

the data to the OrgContacts. The policy AppPolicy2 authorises the OrgContacts to access the internet. However, it enforces that the OrgContacts can connect only to a specific url that can be decided by the user. This is achieved by defined the value aCloudUrl and specifying the sendOnlyTo action in the **have to perform** clause. To make sure the data that is sent are only public contact entries, the **handle** clause is specified for handling only data tagged with ‘‘Public’’ and ‘‘Contacts’’ labels. With this policy, we can prevent the application to send the user’s contact to other online services. Moreover, if the application would get access to other type of data (for example call history) tagged with other labels then this policy would not allow the sending of this data over the internet.

One of the main security issues of the Android platform is the spreading of access permissions. Applications can implement services. These services can be invoked by other applications. In this way, the application implementing the service can allow other applications to use its permission to access phone resources

An application A requests permissionto access the internet. The application A could pose as a simple application to provide news feeds and it would not look suspicious to the user. An application B that acts as a navigation application requires permission to access the GPS to display the user’s current position. Moreover, application B implements a service that allows other applications to access the GPS through application B permission. code that invokes the service of application B to get access to theGPSwithouttheuserknowingaboutthis.Thus,application A without asking the user for the GPS permission would be still able to get that information and it could be able to leak the user’s location over the internet.

We do not expect that the average Android user is able to create policies when installing applications. For this reason, we have extended the Android installer with the User Setting Interface (USI) component (see Figure 1). When a new application is installed, the installer presents to the user the set of permissions that the application requires. These permissions are extracted from the application manifest file. The USI intercepts the permissions requested by an application and generates policies and labels according to type of permissions that the application is requesting. For instance, when the OrgContacts is installed it requires access to the phone contacts (READ_CONTACTS) and internet (INTERNET). The first requirement will trigger the USI to generate a basic policy for OrgContacts to access the contacts and the internet as in Figure3.he USI checks in the Labelling Store for possible extra labels associated with the data type requested from the application. For contacts two extra labels are specified for public and private entries. The USI will prompt the user for allowing the OrgContacts to access these two subtypes. The user selects only the “Public” label to be associated with the OrgContacts. Then the USI will modify the handle clause adding the ‘‘Public’’ label, and generate the have to perform clause for filtering out the contacts tagged with the “Private” label. Moreover, the USI will inform the user that OrgContacts send the contacts over the internet and whether the user is willing to grant this type of permission to the application.

The user will agree to propagate the data that OrgContacts has access to to the internet.

The result of this is shown in Figure4, the user decides that OrgContacts has to access to only specific cloud sevice and selects it for USI . This will allow OrgCOntacts to generate have to perform clause in AppPolicy2 as shown in fig 5

Similarly to the case of the READ_CONTACTS permission, the USI checks whether the user associates data with extra labels for READ_CALENDAR, READ_LOGS, READ_SMS, and GET_ACCOUNTS permission requests. For permissions that enable applications to generate data traffic, such as INTERNET, SEND_SMS, CALL_PHONE and BLUETOOTH_ADMIN, the USI enables the user to set specific destinations or to black list addresses and phone numbers. Moreover, when these permissions are combined with permission for access data then the USI always prompts the user for explicit consent to allow the application to combine the permissions. The user can change at runtime the permissions granted to an application by disabling the respective policy. YAASE enforcement mechanism overwrites the check permission of the standard Android. YAASE enforces a negative- by-default policy meaning that if there is no policy associated with an application request then the request is not granted.

B. Cloud based malware detection

Most of the device malicious behavior detection methods need to operate on the device, however, the device is a resources-constrained platform. Its CPU, memory and power decide that it cannot run complex detections. Its CPU, memory and power decide that it cannot run complex detections.

Before introducing the system, we need to clear security problems faced by the mobile devices, including the following aspects:

- Malicious applications
  The malicious applications are not only found in some third-party application market, also found in the Google Android Market. Very serious problems are caused by these malicious applications. Some applications can reveal the user's location, contacts and other personal information
- Unsafe websites
  Some sites may contain large number of malicious applications. Android phone who using web browser to surf the Internet may be exploited if he/she visits a malicious page.
- Data security of mobile devices
  Smart mobile devices are different from PC because of their portability, and thus they are in a higher risk of loss. When a user lost his/her device, the data on the devices is difficult to recover. Some other reasons such as misuse of users, damage of devices may also cause the loss of data.

 ISSN: 2454-4248
12 – 22

- Network data security of the mobile devices
  When a user connects to an access point setup by the attacker, the traffic of his/her mobile device may be sniffed.

A Cloud based detection system introduced below acts as a good solution to the security issues faced by the android devices.

(1) System architecture :

The system shows that the use of cloud can detect malware on the Android device without the actual intensive-logics running on the device. The architecture of this system is shown in Figure.

An agent application running on the device, which has three basic functionalities; User interface, Operating logic layer and Connection Management. The device firstly sets up a persistent connection with the cloud server. Through the connection, the device uploads suspicious files to the cloud management server and receive results from the cloud server. In the meanwhile, the device connects to the VPN server in the cloud which can route the traffic of devices to the cloud and to ensure the safety of users' network data. A transparent proxy is deployed in the cloud to detect the traffic from application layer.

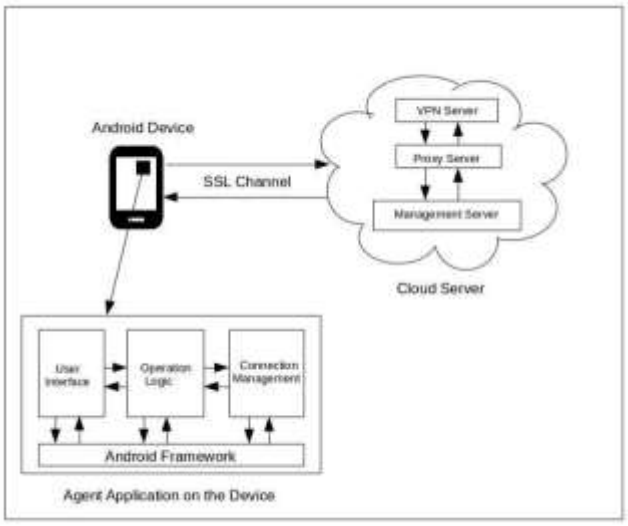

Fig. 9 Architecture of Cloud based security model

(1.1) Agent application on the Device :

(a) User Interface

After entering the user interface, users can select the appropriate action, such as logging, data backup, data synchronization, malicious applications scanning and so on.Then user interface sends commands to the operation logic layer. The logic layer executes the commands and returns the state to the user interface. Then the user interface is updated with the state.

(b) Operation Logic

The operation logic layer contains five components: user validation, application management, device management, information management and data management.

- User Validation - used to initiate a connection to the server and verify the user information.
- Application management - scans the SD card and path /data/app for all the .apk files, extracts the signature of these files and sends them to the cloud. If a malicious application is detected, the cloud will alert the user.
- Device management - users can remotely lock screen, locate the device and back up data through the server when the device is lost.
- Data management - extracts the user's contacts, text messages and other important data, uploads to the server for backup, and restores them when the user needs.
- Message management - responsible for receiving the messages pushed from server with the persistent connection and then notifies the appropriate module for processing.

(c) Connection Management

Mobile devices will connect the cloud server with two connections, one is VPN connection, and the other is persistent connection. The VPN connection can safeguard network data. Due to the VPN connection all the network data of mobile devices will flow through the cloud, so that the server in the cloud can analyze user's network data and protect the devices against threats on Internet.

The cloud cannot send messages to the device directly, as most of the time a mobile device will be allocated a private address and access the Internet through NAT. In order to notify the user immediately about the detected threat the server needs to push the information to the devices. The agent opens a persistent connection to the server for all the information. Every time the server sends a new message to the agent, neither side would close the connection.

(1.2) Cloud Server :

The cloud consists of three servers, namely, VPN server, proxy server and management server. Transparent proxy is made up of VPN Server and proxy server.

- The device first connects to the VPN server. After that, all the network data of the device must go through VPN server. VPN server redirects the traffic to proxy server through netfilter or any other technology.
- Proxy server instead of device requests to the Internet, and detects the content responded from the Internet on application layer. If no threat is detected, proxy returns the content to the client.
- The management server maintains the persistent connections with devices as well as controls the filtering rules of proxy. When threats are detected, the management server notifies users through persistent connections immediately.

In Management components, proxy management sends configuration to the proxy and handles messages sent back from proxy through the Socket. Device management is responsible for notifying devices through persistent connections, receiving data uploaded from devices and processing the data. Log management manages the local log files.

The management server interacts with proxy through socket. The management server sends filtering rules to the proxy. And the proxy sends the suspicious files and applications intercepted to the management server for detecting. The management server interacts with agents on devices through persistent connections. Through persistent connections, the management server can push messages

to devices when threats are detected or the server needs the user to confirm something and device scan send the validation information, apk files, backup data, etc. to server. Presentation layer is provided to administrators. They can check and configure the management server through presentation layer.

(1.3) Measures for safeguard :

In the cloud we implement the following measures to safeguard the safety of devices:

- VPN connections between devices and the cloud ensure the safety of users' network data.
- Deploy a firewall between the Internet and the proxy server.
- Deploy an intrusion detection system on the proxy.
- When a user downloads an application installation package (e.g. .apk file), transparent proxy intercepts the file and sends it to the management server for detecting.
- Management server runs a variety of malicious application detection programs.
- The users can backup their important data on the management server of cloud.

V. CONCLUSION

This paper gives a brief overview of the Android framework and its components namely Activity, Content Provider, Broadcast Receivers and Service. It discusses the possibility of various attacks and risks that can be caused due to these components. We also discuss about the Android malware attacks and their solutions using static and dynamic analysis while giving a secure banking application example. A crucial proposed framework defending these attacks that we emphasize on, is that of YAASE. We also confer upon cloud based malware detection techniques and how they can prevent the user and applications from malicious attacks and how they can help in reducing the load on systems.

REFERENCES

[1] Lucas Davi, Alexandra Dmitrienko, Ahmad-Reza Sadeghi, and MarcelWinandy. Privilege escalation attacks on android. In Proceedings of the13th international conference on Information security, ISC'10, pages 346–360, Berlin, Heidelberg, 2011. Springer-Verlag.

[2] Michael Dietz, Shashi Shekhar, Yuliy Pisetsky, Anhei Shu, and DanS.Wallach. Quire: Lightweight provenance for smart phone operatingsystems. In 20th USENIX Security Symposium, 2011.

[3] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox,Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. TaintDroid:an information-flow tracking system for realtime privacy monitoringon smartphones. In Proceedings of OSDI 2010, October 2010.

[4] William Enck, Machigar Ongtang, and Patrick McDaniel.On lightweight mobile phone application certification. In Proc. CCS'09, pages 235–245, 2009

[5] IDC Worldwide Mobile Phone Tracker, http://www.idckorea.com/, 2014

[6] Seung-Hyun Seo, Dong Guen Lee, Kangbin Yim, "Analysis on Maliciousness for mobile application", IMIS 2012, pp. 126-129, 2012.7.

[7] ZDNet Korea, http://www.zdnet.co.kr/news/news_view.asp?Artice_id= 20120517094852&type=xml, 2012.5.

[8] Taenam Cho and Nammee Moon, "Smartphone Application Development and Code-sig- ning," KIISC, vol.21 no.1, pp.19-25, 2011.

[9] AndroidManifest, http://developer.android.com/guide/topics/ manifest/ manifestt-intro.html

[10] Android Security, http://developer.android.com/training / articles/security-tips.html

[11] ContentProvider, http://developer.android.com/intl/ko/reference/android /content/ContentProvider.html.

[12] Android Permission, https://android.googlesource.com/platform/frameworks /base/+/master/core/res/AndroidManifest.xml

[13] Dalvik, http://ko.wikipedia.org/wiki/%EB%8B%AC%EB%B9%85_(%EC%86%8C%ED%94%84%ED%8A%B8%EC%9B%A8%EC%96%B4.

[14] Smali, http://code.google.com/p/smali/.

[15] Android Proguard, http://developer.android.com/tools /help/proguard.html.

[16] Android API Guide - Permission, http://developer.android.com guide/topics/manifest/permissionelement.html.

[17] Laurence Goasduff. Gartner Says Worldwide Smartphone Sales Soared in Fourth Quarter of 2011 With 47 Percent Growth. http://www.gartner.com/it /page.jsp?id=1924314.

[18] W. Enck, M. Ongtang, and P. McDaniel. Mitigating Android Software Misuse Before It Happens. Technical Report NAS-TR-0094-2008, Network and Security Research Center, Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA, November 2008.

[19] Thomas Blasing, Aubrey-Derrick Schmidt, Leonid Batyuk, Seyit A.Camtepe, and Sahin Albayrak. An android application sandbox system for suspicious software detection. 5th International Conference on Malicious and Unwanted Software (Malware 2010), Nancy, France,2010.

[20] Georgios Portokalidis , Philip Homburg , Kostas Anagnostakis , Herbert Bos, Paranoid Android: versatile protection for smartphones. 26th Annual Computer Security Applications Conference, December 06-10, 2010, Austin, Texas

[21] Amir Houmansadr, Saman A. Zonouz, and Robin Berthier. A Cloud-based Intrusion Detection and Response System for Mobile Phones. IEEE/IFIP 41st International Conference , 2011.

[22] R. Naraine. Google Android vulnerable to drive-by browser exploit. http://blogs.zdnet.com/security/?p=2067, October 2008.

[23] Alex Tsow. Phishing with consumer electronics - malicious home routers. 15th International World Wide Web Conference (WWW2006), May 2006.

 ISSN: 2454-4248
12 – 22

[24] J. Oberheide, E. Cooke, and F. Jahanian. CloudAV: Nversion antivirus in the network cloud. 17th USENIX Security Symposium, San Jose, CA, July 2008.

[25] J. Oberheide, K. Veeraraghavan, E. Cooke, J. Flinn, and F.Jahanian. Virtualized in-cloud security services for mobile devices. MobiVirt '08, pages 31–35, June 2008.

[26] Malicious apps hosted on Google store turn android phone into zombies http://arstechnica.com/gadgets/2012/05/malicious-apps-hostedin-google-market-turn-android-phones-into-zombies/

[27] Google now scanning for android apps for malware http://www.cnet.com/news/google-now-scanning-android-apps-formalware/

[28] Felt, Adrienne porte; Chin, Erika; Hanna, Steve; Song, Dawn;Wanger, David. Android Permissions Demystified 2012, http://www.cs.berekeley.edu/~afelt/aandroid_permissions.pdf