International Journal of All Research Education and Scientific Methods (IJARESM)
ISSN: 2455-6211, Volume 3, Issue 4, April 2015

Vulnerable Android: A Study on UI Inference Attacks and Malware Attacks

Arun Sharma[1], Harmeet Malhotra[2]

[1]Research Scholar, Institute of Information Technology & Management, (Affiliated to GGSIPU), New Delhi
[2]Associate Professor, Institute of Information Technology & Management, (Affiliated to GGSIPU), New Delhi, India

ABSTRACT

Smartphone industry has been booming with phenomenal growth with the discovery of new technologies. Over the past two years, the rapid growth in smartphones and tablets usage has led to inevitable rise in targeting these devices by the attackers. Although various security mechanisms are added in the consecutive versions of Android operating system, but attackers are still able to exploit them. The exponential growth of Android OS due to adoption by various manufacturers and largely unregulated android app market produce a sharp rise in security threats targeting that platform. In this paper, we discuss a new type of attack that directly breaks the GUI integrity of the operating system and breaches the UI state of any application. It does not require any special permission and can easily be implemented by the attacker. This type of attack is called UI state inference attack. We discuss its design and demonstrate this attack by successfully hijacking the UI State of a widely popular application. This paper also discusses about android malwares which usually run in the background and are not only used to steal sensitive information from the user but also able to avoid and counter detection methods. We analyze various android spywares and design summary which are available to anybody on the web. Nowadays, attackers have developed methods to counter detection methods by having full control over the system. Zero-day vulnerabilities are used to get the root access and manage the system by binding rootkits with the malicious applications. This paper discusses their consequences and suggests various mitigation strategies to avoid them.

Keywords: Android, Inference Attacks, Malware, Rootkits, Spywares.

INTRODUCTION

Currently, Google's Operating system Android holds the biggest market share among all Smartphone operating systems (as shown in Fig 1). Therefore, there must be efficient security mechanisms to protect user from attackers. Since Android has very few restrictions for the developer, thus increases the security risks for the end users. One of the reasons that Android has succeeded in gaining major market share is that it is an Open Source; hence, free for manufacturers to implement (excluding patent settlements). This leads to substantial fragmentation of Android versions between the devices and means that vendors are reluctant to roll-out updates to their devices, presumably out of some concern regarding driving demand for future devices. The security patches and updates that rolls out after discovering vulnerabilities in the versions are thus not supplied to end users timely which increases the security risks for users. Google has tried to encourage manufacturers to update their devices to the latest version but been largely unsuccessful in doing so. As a result, vulnerabilities are left unpatched in manufacturer's stock ROMs and advanced users (who has root access to the device) are turning to flash custom ROMs that might be incompatible with the devices which leads to whole lot of other issues.

Android is the victim of its own success, its open source nature and very few restrictions on developers result in different types of malicious attacks. Some attacks are that of malwares i.e. those abuse the permissions API of Android operating system and perform malicious operation in the background without the knowledge of the user. There are also spywares those collect as much private information as possible from the users. These types of spywares can silently monitor devices via web browser, calls, text messages, photos, videos, GPS locations, and even visited URLs from the device. Since the Google Play store are weakly moderated by the management, these types of malwares are uploaded on the play store by the attackers and downloaded by billions of people all over the globe. Some of the most dangerous Android malware attacks are:

- **Fake Banking Apps:** These lure the users/customers into entering their login details for any of their online accounts.
- **DroidDream:** It has infected the devices, then breached the android security sandbox and finally stole the data.
- **AndroidOS fake player:** It is a kind of a media player that silently sends SMS to premium SMS numbers.

Also, there are other attacks which have been discovered recently, that breaks the GUI integrity and confidentiality of the Android operating system leading to serious security consequences, UI inference attacks. In this attack, attacker first builds the UI state machine which is constructed offline with the help of shared memory channels and breaks GUI integrity by carefully exploiting the designed functionality that allows UI preemption, which is mostly used by reminder apps or alarms on Android. Some of the UI state inference attacks are:

- **UI state being hijacked** for stealing confidential user input, for example, login credentials
- **Obtaining sensitive camera images shot by users**, for example, personal check photos for banking apps
- **Inferring user behavior** by tracking UI state changes
- **Enhancing existing attacks in both stealthiest and efficiency** by providing the target UI state

In this paper, we report that how these types of attacks are implemented, how they work and what should be done to prevent them. Specifically, we are going to study about UI (Activity) Inference Attacks and Android malwares that breaks the integrity and can be used to steal sensitive data from the user. We provide relevant background information and overview about how an attacker gathers information to perform the attack and can use that information to steal data from the user. The Design details of the attack are briefly described and explained. We also suggest some of the mitigation strategies for these attacks.

UI STATE INFERENCE ATTACKS

In **UI state inference attack**, the attacker first makes an UI interface based on the UI states of the victim app and then infers that interface in real time while the user is navigating in the victim app. It just pops up the attacker interface in place of victim app's interface while monitoring it from the unprivileged background service of the attacker's application. The UI state is commonly known as Activity in Android Terminology. Thus, we generally call it Activity inference attack. The inference attack requires **no Android permission**.

THREAT MODEL

UI State Inference Attack's Threat Model requires the attacker app running as the background service in the victim's device which is obviously required for any attack. To ensure from recognizing this background activity by the user, the attacker's app should have low-overhead and should not be draining the battery so quickly. Also, the attacker's app should not have extra permissions beside INTERNET permission for sending data to the attacker. Activity Inference is done in two steps as show in Fig. 1.

Steps:

1. **Activity transition detection:** Activity transition event is detected by reporting a single bit of information to know whether an Activity transition just occurred. This is enabled by the newly-discovered shared-memory side channel. The change observed through this channel is a highly-distinct "signal".

2. **Activity inference:** After detecting an Activity transition, we need to know which activity is acting as the foreground activity. To do so, we design techniques to train the "signature" for the landing Activity, which roughly characterizes its starting behavior through various publicly observable channels such as CPU utilization time, network activity and shared-memory side channel.

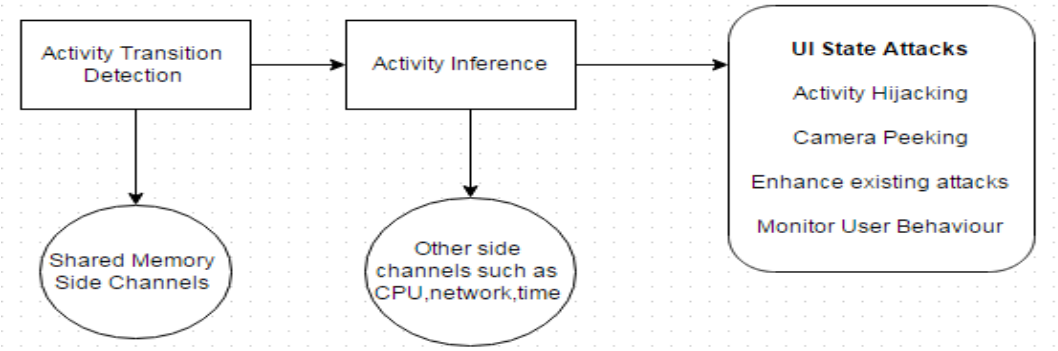

Figure 1: Activity Inference Attack

Finally, after getting the information about the activity transition and foreground activity in real time, we can develop various novel attacks that can perform various actions such as stealing sensitive data, camera hijacking etc. For example, the attacker app lies in the background waiting and monitoring for the target app to open the

required activity. At exactly the right moment, the attacker app interferes with the target app and hijacks the activity to perform various types of attacks to steal user's data.

ANDROID ACTIVITY AND ACTIVITY TRANSITION

This attack is based on the UI states of the Activities. An activity is a component that provides screen which users can interact with to do something. An Android application is basically a set of activities in which each activity is responsible for certain action. The user goes from Activity A to Activity B to Activity C to do certain task and so on. Due to security concerns, one cannot know which activity is in foreground unless they are the owners or they have access to central Activity Manager. An Activity maintains a View State once it is created. After the activity transition took place, these activity view states can be retained.

Activity Transition

An application usually consist set of multiple activities that are loosely bound to each other. Activity A can be transitioned to Activity B to support the functionality of the app. When the current activity initiates another activity, then this new activity is pushed onto the top of the stack and takes the focus. The previous activity is stopped but it remains in the stack. Whenever an activity stops, the system maintains the current state of its user interface. Next, when the user presses the *Back* button, the latest and current activity is popped from the top of the stack (the activity is destroyed) and the previous activity resumes from where it has been stopped i.e., the previous state of its UI is restored. The activities stored in the stack are never rearranged, rather they are only pushed and popped from the stack—pushed onto the stack when started by the current activity and popped off when the user leaves it using the *Back* button. The back stack operates as such in the form of "last in, first out" object structure. Fig. 2 shows the timeline of the activities on the back stack at each point of time.

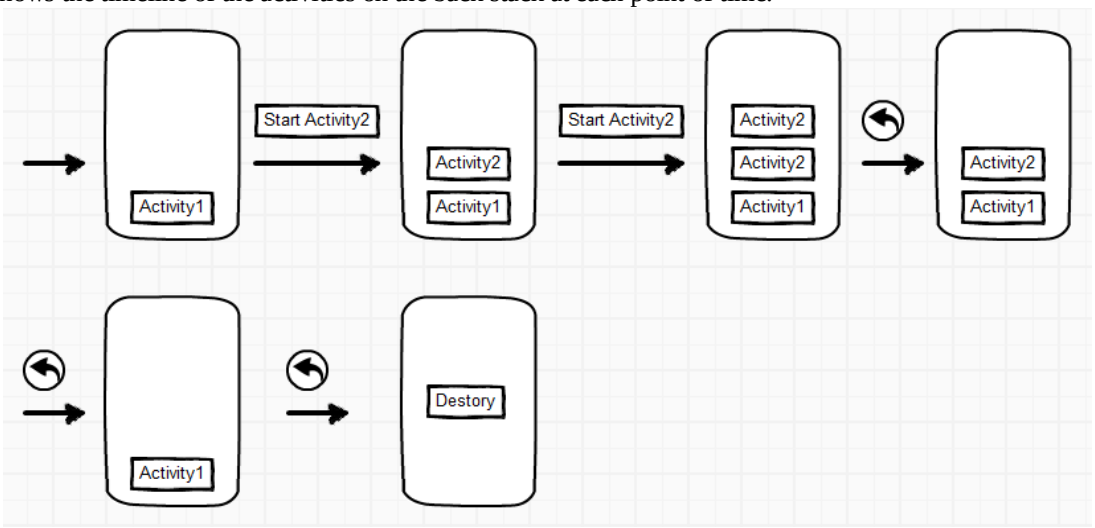

Figure 2: A representation of how activities are added to the back stack

During launch, the create transition calls both onCreate() and onResume() in which onResume() method is called when the activity is resumed. Both onCreate() and onResume() are implemented by the app. After that, performTraversal() is called, in which measure() and layout() calculate the sizes and locations of UI components, and draw()puts them into a data structure as the new Activity UI. After Activity UI states is created, create transition stops the current activity and puts it on Back Stack (as shown in Fig 3).

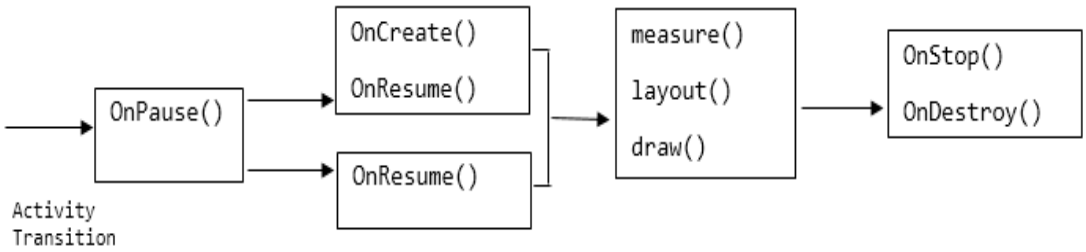

Figure 3: Different function calls involved in activity transition

ACTIVITY INFERENCE

After detecting an Activity transition, we can know about the identity of foreground activity by two kinds of information:

1. Activity signature: onCreate() and onResume() which are involved in the activity transition, are defined in the landing Activity, and the execution of performTraversal() depends on the UI elements and layout in its LandingState. Thus, every transition has behavior specific to the landing Activity, giving us opportunities to collect information about the foreground activity based on the data collected during the transition.

2. Activity transition graph: Once the foreground activity is known, if the Activity transition graph of the target app is sparse, then the set of next candidate activities become limited which further ease the difficulty in the inference.

EXAMPLE ATTACKS: ACTIVITY HIJACKING

Activity Hi-jacking is a new type of Android attack in which attacker can gain the sensitive information by breaching the GUI integrity of the target device. As the name suggests, it stealthily hijacks the foreground activity in the target app and inserts its own custom phishing activity right before when the user is about to navigate to the target activity. The user then enters sensitive information in the malicious activity which is then sent to the attacker in the background. There are many apps available on the play store which monitor the device in the background and preempt the foreground activity to perform some action. For example, AppLock is an application in which user gets the lock screen when user is about to open the locked application. In this way, attacker preempts the foreground activity and puts the fake activity in the front which looks similar to the original activity and thus, hijacks user to enter sensitive information in the Fake activity.

ACTIVITY HIJACKING ATTACK OVERVIEW

As shown in Fig. 4, Activity Hi-jacking occurs in three steps as follows:

Step 1: Attacker's app has a background service running which monitors the foreground activity of the target app. It waits for a particular activity such as LoginActivity to come to foreground.

Step 2: Once the target app's activity is about to enter the foreground, the attacker's app simultaneously injects its own pre-prepared phishing LoginActivity which has same design as that of target app into the foreground. Note that the challenge here is that it introduces a race condition whether the phishing activity comes too early or late which will lead to visual disruptions or some glitches during navigation which could be noticed by the user. With carefully designed timings and disabling animations, it can be eliminated. When the user enters sensitive information such as credentials into phishing login activity, it will be sent to attacker at some remote location.

Step 3: After receiving the sensitive information, the attacker's app ends the attack unsuspicious. For example, displaying incorrect login credentials error details. In the meantime, it transitions back to original app making user to believe that he/she might have entered the wrong credentials.

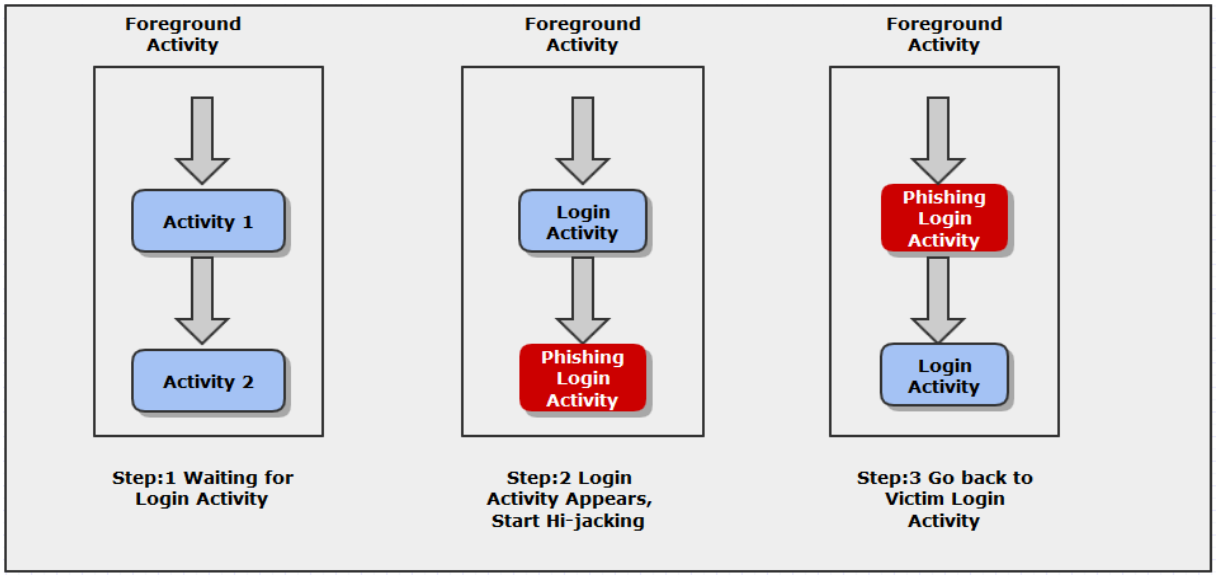

Figure 4: Steps of Activity Hijacking Attack

MITIGATION STRATEGIES

Improving access control strategy for Proc file system

UI Inference attack is based on the freely accessible files in the proc file system. Shared virtual memory and features of activity signature are dependent entirely on these files. Simply removing them from publicly accessible files is not an option because of large amount of existing apps dependent on them. To better solve this problem, we can reduce the attack effectiveness by reconsidering access control strategy for these public accessible files to balance functionality and security. In fact, Android has already restricted access to certain proc files that are publicly accessible in the standard Linux e.g /proc/pid/smaps.

Reusing Window Buffer

The Activity transition signal consists of detecting change in the shared virtual memory corresponding to the windows buffer allocation and de-allocation. To eliminate such signal, systems can avoid them by pre-allocating two copies of the windows buffer and reuse them for all transitions in the app. But, this will increase the memory usage for each app as buffers are several MB in size. Since, larger memory size will be available in the future mobile devices, this strategy might be acceptable.

Enforcing UI State transition animation

UI state transition animation plays an important role in informing users about the states of the app. In the Activity Hijacking attack, the animation is explicitly turned off by the attacker before activity injection so that it does not look suspicious to the user. Thus, one defense is to always keep the animation indicator on by enforcing animation in all UI state transitions. This can help a lot in reducing the attacker's app efficiency.

Limit Background Functionalities

Since background applications do not directly interact with users, they cannot be used to perform privacy sensitive operations. In Camera Peeking attack, the preview frames are captured in the background without the knowledge of the user. Thus, Android must impose more restrictions on the use of background related operations such as restricted application to use sensitive resources such as camera, GPS, sensors etc.

ANDROID MALWARES

Malware is malicious software used to steal sensitive information from user, or gain access to private systems. It is a type of intrusive software or application that can appear in the form any script, executable code or any other form. Nowadays, Mobile phones have become the victim of malware attacks. Amongst the all mobile phone malware attacks, the largely targeted by the hackers are the Android smart phones. The reason is that the Android applications market provides an open platform for the applications. Second major factor is that more than fifty mobile phone companies manufacture smartphones with the Android operating system.

THREAT MODEL

Every Malware is different than the other. But the main motive of them is to extract the information from the user and send it to the attacker. Malware can collect privacy-sensitive information by requesting certain permissions. Attacker can use these permissions to perform any malicious tasks like sending your location, sending SMS to any number or can even use root-exploit kits to perform system-level tasks to gain more access to your device.

Android malware has evolved from a simple Trojan that sends SMS to premium-rate services, without the knowledge and authorization of the user to sophisticated code that is able to infect legitimate applications, encrypt data using symmetric-key encryption (AES), propagate via the official Android Market, get root privileges in the device using two different exploits, receive and execute commands remotely, act as a downloader by installing applications without user's knowledge and, lastly, dynamically load a payload from a remote server.

Here's how it works: First, a person downloads a malicious app. This app then invisibly sends a text message to a service that uses a middleman service having a relationship with the malware author. Next a message giving the confirmation is sent back to the malware, which blocks it from so that it cannot be seen by the customer and confirms the charge finally. The charge adds to the user's bill, and the carrier takes its cut and gives the remaining amount of money to the service and the middleman, and thus to the malware author (shown in Fig. 5).

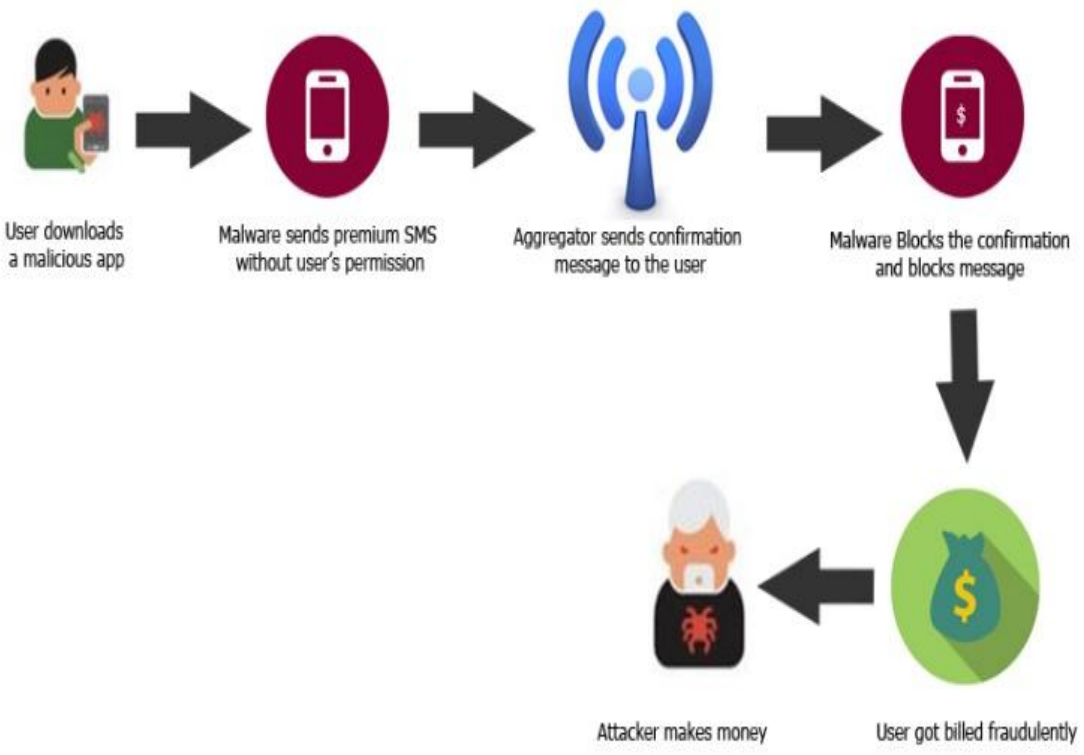

Figure 5: working of typical malware that abuses READ_SMS permission

Example : Rootkits

Most malicious programs are detected by the antivirus applications which monitor the system periodically for malicious access. But in recent years, attackers are developing special programs "Rootkits" that are designed to maintain privileged access to the system through hiding certain process and redirection of system calls. These rootkits are bundled with the malware to prevent detection by the antivirus programs. Once installed, it is not only used to hide the intrusion but also attacker has full control over the system. It means that other application's data can be modified including the detection programs.

While other malwares such as spywares abuse the sensitive permissions, rootkits are used to exploit the sandbox environment by gaining privileged access to the system. It means that any malicious application with root access can create, modify or delete any file in the system. The rootkits are used to maintain that root access in a better way as it is often difficult to maintain the root privileges because of the "noise" (such as logs, running processes) that may have been created while gaining root access. Since Android is based on the Linux kernel, attacker can gain administrative or root access through existing vulnerabilities (or zero-day vulnerabilities) that might exists in Linux OS.

Rootkits have different flavors ranging from highest privileged access to least privileged variants: user mode, kernel mode, firmware and hypervisor. The most popular and widely used on Android is user-mode and kernel-mode rootkit. The user-mode rootkits modify the existing classes in Dalvik VM libraries. For example, the attacker can swap *WebView* class (which is web browser class) with a custom-made for monitoring browser's activities. Kernel-mode rootkits are lethal and have highest privileges as the operating system making it resistant to the anti-malware programs as they can easily manipulate the data being analyzed.

ROOTKITS: DESIGN DETAILS

Bypassing Permission System

Android permission model is majorly affected by the rootkits. As one application can only access the resources which are specified in the manifest file of the application. However, there are some special permissions that are only accessible by system applications that are installed in system partition such as INSTALL_PACKAGE permission which allows application to install another application without any confirmation. These permissions are not accessible to normal applications. But by having root access and using rootkits, they can bypass permission mechanism as all permissions are permitted to the system applications including applications running with root privileges. One of the widely popular rootkit which was discovered is GingerMaster. It is packaged with root exploit and installs the malicious application in the system partition by bypassing permission mechanism. It

is not only used to steal sensitive information but also contains a payload to connect to the remote server and install more malwares on the system.

Hooking Activity Priorities

Activity is the basic element of the Android operating system that is used to display user-interfaces to the user. Each activity can be associated with any action which is specified by intent-filters in the manifest file. For example, android.intent.action.MAIN action in Intent filter is used to specify which activity will be launched when the application icon is tapped. There are priority values associated with Intent-filters which determine which activity should be handled when there are multiple activities. If an attacker has access over these priority values, the lowest priority valued activities can be hidden. This is threatening as priorities associated with particular activities can never be larger than the default value unless it is a system application. If attacker has root access, then malicious app can be installed as the system application. Thus, attacker with root access has full control over activity priorities in the system.

MITIGATION STRATEGIES

Improvising Trusted Application Stores

Most trusted app store for android is Play Store managed by Google. There are thousands of applications uploaded by developers all over the world daily. App stores must develop some programs to detect malwares in the applications and prevent them from reaching to billions of users. There are reports which states that many malware applications are able to bypass app store's malware detection programs and able to reach millions of users. Therefore, effective detection tools must be used to detect malwares on the application stores.

Vendor Specific Protection

Android device vendors must add some security mechanism to prevent system software from overwriting. The main reason for this is to prevent attackers from gaining control over the system. NAND Lock can be used to block access which is implemented as kernel-mode driver. It blocks all write requests to important partitions such as system partition. Security Frameworks such as LSM (Linux security modules) can be used to prevent modification of the system.

Inbuilt Detection Tool

Android must provide an in-built malware detection tool that can be used to monitor all applications across the system including system applications periodically. This in-built malware tool should not be installed on the device as it can be modified by the applications with root access. These programs must run as services and directly connected to internet for checking latest vulnerabilities and notify the user as soon as they detect.

CONCLUSION

Since android is an open-source operating system, it leads to lot of security consequences associated with it. Google has implemented sophisticated security model and environment to protect user's privacy, but sooner or later, attackers are able to exploit it. In this paper, we have shown that how much vulnerable Android operating system is. We have studied a new type of attack which is recently discovered and is very dangerous – UI inference attack that is not dependent on Android security mechanisms. It directly breaks the GUI integrity of the operating system. It doesn't require any additional permissions. Other operating systems are also susceptible to these UI inference attacks. We have suggested various mechanisms to mitigate these UI state attacks. Android APIs are very much vulnerable to these attacks and there is need to put some restrictions on the usage of some resources which are used by the attacker in the background. We have also studied Android malwares which have been increasing at the growth rate of Android operating system. With new advancements, attackers are able to bypass Android security mechanisms and able to steal sensitive information from the users. Spywares and Rootkits are currently used by attackers for monitoring and full control on the system. These malwares are not only used to steal user's privacy, but also have abilities to counter detection methods and have payload to download more malwares into the system. We have also discussed strategies for eliminating malwares and suggested strategies to fight them.

REFERENCES

[1] Luyi Xing, Xiaorui Pan, Rui Wang, Kan Yuan and XiaoFeng Wang, "Upgrading Your Android, Elevating My Malware:Privilege Escalation Through Mobile OS Updating".

[2] William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri ,"A Study of Android Application Security".
[3] Chit La Pyae Myo Hein , "Permission Based Malware Protection Model for Android Application,"
[4] "Android Camera," Available [Online] http://developer.android.com/ reference/android/hardware/Camera.html.
[5] S. Chen, J. Meseguer, R. Sasse, H. J. Wang, and Y.-M. Wang, "A Systematic Approach to Uncover SecurityFlaws in GUI Logic," in IEEE Symposium on Security and Privacy, 2007.
[6] S.Jana and V. Shmatikov, "Memento: Learning Secrets from Process Footprints," in IEEE Symposium onSecurity and Privacy, 2012.
[7] Y. Zhou and X. Jiang, "Detecting Passive Content Leaks and Pollution in Android Applications", Proceedings of the 20th Network and Distributed System Security Symposium, NDSS, February 2013.
[8] "Transparent Activity Theme," Available [Online] http://developer.android.com/guide/topics/ ui/themes.html# ApplyATheme.
[9] P. McDaniel, and A. Sheth, "TaintDroid: An Information Flow Tracking System for Real-Time Privacy Monitoring on Smartphones", 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI'10), October 2010.
[10] Sophos Mobile Security Report 2014, Sophos Labs, Available [Online] http://www.sophos.com/en-us/medialibrary/PDFs/other/sophos-security-threat-report-2014.pdf
[11] Trends for 2013:Astounding growth of mobile malware, ESET Labs, Available [Online] http://www.eset.com/us/resources/white-papers/Trends_for_2013_preview.pdf
[12] Mobile Malware and Spyware: Working Through the Bugs, NIST Forensics, July 2014 Available [Online] http://www.sans.org/event/san-francisco-2014/bonus-sessions/4527
[13] Available [Online] Permission Tree. http://developer.android.com/guide/topics/ manifest/permission-tree-element.html
[14] "Android Broadcast Receiver," Available [Online] http://developer.android.com/reference/android/content/BroadcastReceiver.html.
[15] GingerMaster: First Android Malware Utilizing a Root Exploit on Android 2.3 (Gingerbread), http://www.csc.ncsu.edu/faculty/jiang/GingerMaster
[16] C.-C. Lin, H. Li, X. Zhou, and X. Wang, "Screenmilker: How to Milk Your Android Screen for Secrets," in NDSS, 2014.