Methods for avoiding rooting in Android System
Abstract

Android Rooting allows the user to modify the system such as removing the pre-installed apps they dislike. However, rooting is not easy like before. Google has introduced SELinux to the Android system. It required doing more work to achieve rooting function. Unluckily, some serious Android root exploits made Google’s beautiful plan wasted. One-click root apps or software used these exploits to root all the versions through Android 5.1.1. Then, Google strengthened control of system partition from Android 6. But there came out new method required an unlocked bootloader to root the devices. Hiit Media wants me to think out some solution to avoid rooting. After investigating the cause and effect of the rooting mechanism, I provide them the solutions in the end.

Keywords: Android root exploits; SEAndroid Rooting
Preface

I would like to thank my supervisor Morgan Ericsson. He took time to have the meeting with me about the problem and direction of my thesis. Also, he took time to review my work during the busy business trip to Argentina. And I would like to thank my other two external supervisors from Hiit Media. Janne Jöhag answered me some questions by email patiently. Ulf Pettersson even prepared a presentation to show me the company’s vision and provide me with demo tablet for testing. In addition, I would like to thank Ola Flygt who is my Program Coordinator. He helped me a lot during the Bachelor study and then I had the chance and ability to start Bachelor thesis work. In the end, I would like to thank my parents who support my study in Sweden, which would change my life.
Contents

1 Introduction .......................... 1
  1.1 Background .......................... 1
  1.2 Previous research ..................... 1
  1.3 Problem formulation .................. 1
  1.4 Motivation .......................... 2
  1.5 Research Question .................... 2
  1.6 Scope/Limitation ..................... 2
  1.7 Target group ........................ 2
  1.8 Outline .............................. 3

2 Method ............................. 4
  2.1 Scientific approach ................... 4
  2.2 Method description .................... 4
  2.3 Reliability and Validity ............... 4
  2.4 Ethical considerations ................. 4

3 Testing Device and System .......... 5
  3.1 Samsung Google Nexus 10 .............. 5
     3.1.1 Google Nexus 10 Stock Android 4.4.4 ............. 5
     3.1.2 Google Nexus 10 Stock Android 5.1.1 ............. 5
     3.1.3 Company’s pre-configured AOSP system Android 4.4.4 ............. 6
  3.2 Huawei Google Nexus 6p ............... 7
     3.2.1 Google Nexus 6p Stock Android 6.0.1 ............. 7
     3.2.2 Google Nexus 6p Stock Android 7.1.2 ............. 7
     3.2.3 Google Nexus 6p Beta Android O ................. 8
  3.3 Samsung S8 .......................... 9
     3.3.1 Samsung Galaxy S8 Stock Android 7.0 ............. 9

4 SEAndroid .......................... 11
  4.1 Android Application Sandbox .......... 11
     4.1.1 Brief introduction of sandbox model .......... 11
     4.1.2 Sandbox model in Android ................. 12
  4.2 SEAndroid Introduction ............... 13
  4.3 How to root in SEAndroid ............. 14

5 Taxonomy of Android Rooting Methods .. 16
  5.1 System root .......................... 16
     5.1.1 Android Root Exploit Impact and Coverage .... 16
     5.1.2 Methods .......................... 24
  5.2 Systemless root ...................... 26

6 Penetration tests for different Android Versions .................. 28
  6.1 Root Checker .......................... 28
  6.2 One-click root software .............. 28
  6.3 su binaries .......................... 28
  6.4 Result ............................... 29

7 Samsung Knox ........................ 34
1 Introduction

The company Hiit Media desires to develop a free Android tablet to distribute to all families in Sweden. This tablet uses cellular network or Wi-Fi to receive the advertisements, weather alerts, emergency notices or personal invoice information. The company makes money by advertisement. Android Root permission is the highest authority in the system. It can modify anything including deleting some system-build functions or apps, such as company’s advertisement components. They do not want the customers to use root to get rid of advertisements with a free tablet. My purpose is to investigate Android security mechanism and solve the problem for the company.

1.1 Background

Android is the mobile operating system based on Linux kernel and developed by Google. It is known for its open-source development. From Android Open Source Project (AOSP), different companies, groups or single person can modify their own functional operating system in mobile devices, such as Hiit Media’s free Android tablet. Android system also uses Linux permissions and file-system ownership. The root is the superuser of the Linux. The users would like to use root to remove some unnecessary system applications or set up the interface of the Android system. Moreover, with the root permission, it can close the self-starting processes to save the electricity of the devices. However, this action also leads to the damage and security of the system. Google has increased control of rooting version by version including cooperating with NSA (National Security Agency) to develop the SEAndroid based on SELinux. Still, there are possible methods to root the device according to the Android’s versions.

1.2 Previous research

The researchers from the University of California, Riverside, wrote an article called Android Root and its Providers: A Double-Edged Sword in 2015[1]. They described the origin of the root exploits and exploit requirements. And they also checked the Anti-Virus apps whether they would detect the root exploits. In the end, they analyzed the root providers about their performance and protection mechanisms. In my paper, I would cover the origin of the root exploits again with the latest experience, since the previous paper was from 2015 but the techniques change fast. Also, I would try the root providers to see whether they could work in latest Android version. Moreover, the researchers from the University of British Columbia wrote an article called Android Rooting: Methods, Detection, and Evasion in 2015 [2]. They described the root binary implemented mechanism and the detection of rooting. In my paper, I would cover the root binary implemented mechanism again with the latest experience, since it has been another three new versions of Android released from two years ago. There are other researches related to the subject. But these two have the good structure and provide the overview of rooting from theory to practice.

1.3 Problem formulation

The company would use the Android version 4.4.4 which is four years old. Although Google implemented the SEAndroid Enforcing mode to it, there still exists plenty of vulnerabilities. They can be used as root exploits, such as well-known TowelRoot. The root tools could use root vulnerabilities to get the root access of device. The latest version is
Android 7.1.2 and beta version Android O which could be Android 8.0. The new version of Android could fix the vulnerabilities and make the rooting method inactive directly. It is necessary to investigate the differences among Android versions in vulnerabilities. Also, there are other methods to root the device without using the vulnerabilities. It is necessary to find out the mechanism of them and the methods to prevent them. The company requires solutions for preventing rooting in customer’s tablets. They knew that there are always some hackers who would try hard to root the device. The company only requires the methods to avoid easy rooting for normal customers. Firstly, it is necessary to find parameters to measure the advantages or disadvantages of the solutions. Then, by comparing different solutions with the aspects above, there should be the best option for the company. The report aims to find the solutions that balance between effectiveness and cost.

1.4 Motivation

Rooting is not the problem only related to this company’s income advertisement. Rooting is also a significant security issue in Android system. If you wish to have a secure device, you must take care of the root permission. Finding the solution for avoiding rooting is the key for the system integrity. The solutions will be investigated from hardware and software parts of computer science area. The solutions should be refreshed in the academic area, and company needs with current technologies. The results in the paper could be the reference when other companies would like to deploy a safer Android platform.

1.5 Research Question

<table>
<thead>
<tr>
<th>RQ1</th>
<th>What solutions exist to prevent rooting in Android System?</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ2</td>
<td>What parameters exist to measure the solutions above?</td>
</tr>
<tr>
<td>RQ3</td>
<td>Which technology is the most suitable solution by using the parameters above for the company?</td>
</tr>
</tbody>
</table>

1.6 Scope/Limitation

Android system has a feature which is fragmentation. Different manufacturers almost have various Linux kernel and Android versions. The fragmentation can be an advantage that manufacturers can modify the devices according their special functions with specific hardware. That is why the Android environment is colourful and diverse. But it could also be the disadvantage that vectors cannot fix the vulnerabilities in time and deploy the latest Android version even after one or two years. I only have several devices for testing. Company provides me with the Google Nexus 10 which can test their own AOSP system, Google stock Android 4.4.4 and Google stock Android 5.1.1. I also get my own Google Nexus 6p which can test Google stock Android 6.0.1, Android 7.1.2 and latest beta Android O. In addition, I get Samsung flagship galaxy S8 for some tests as a third manufacturer device example instead of Google. Google’s devices like to keep and modify one version kernel among different Android version. As a result, some of my paper content may not apply to other manufacturers’ devices.

1.7 Target group

Some mobile phone company’s security experts would be interested to see which Android version or policy is suitable for their groups. In addition, my paper would help later
researchers to investigate the advance of rooting mechanism.

1.8 Outline

Chapter 3 will describe the test device for my paper including the Android test environment for each instrument. Also in this chapter, there are some limited parameters for the different device for RQ2. For example, Google does not support the Android version above 5.1.1 in Nexus 10.

For RQ1, if I want to find out the solutions, I need to understand the mechanism of rooting. SEAndroid is a critical security component since Android 4.4. Chapter 4 will explain Android runtime mechanism to show how it affects the rooting process and how to root in it. Later, chapter 5 will describe the taxonomy of Android Rooting Methods, system and systemless root. In system root part, it will also discover the Android root exploits which are used in the rooting process when the device’s bootloader is locked. It includes four categories root exploits and the corresponding vulnerabilities detection in different Android version. Then it comes to the methods which includes the relationships among boot sequence, temporary root access mechanisms and su installation packages.

Now we should know the mechanism of rooting. To show the different behavior in different Android versions, I need to do the penetration test against them. Chapter 6 will describe the actual operation to gain the root in my testing devices. It shows different results when using different rooting methods. As a part of the solution, detection of rooting is necessary. Samsung has provided a hardware-based security method which would detect the rooting. Chapter 7 will describe the Samsung Knox for device security. It has a mechanism to identify the unlocking bootloader and initiate safety measures. But on the software side, Google has provided an excellent API to detect the rooting. Chapter 8 will describe Google’s SafetyNet API which can identify whether the device firmware has been changed (system partition) and has a known Android compatibility profile match on Google server.

Chapter 9 concludes the results from the investigation above and get the solution and measuring parameters. Chapter 10 analysis the solutions above and provide company most suitable solution according to the metrics.
2 Method

This chapter would describe the scientific approach that would be used to answer my research problems.

2.1 Scientific approach

To find the possible solutions for RQ1, I need to do the rooting against different Android systems step by step. I have three Android devices for testing, Nexus 10, Nexus 6p and Samsung Galaxy S8. I have installed the available test Android system one by one in each of the device. And then I run specific checking app for scanning vulnerabilities and did the penetration test to root the device with different approaches. These methods above are quantitative. Then I would describe the user interface reflection in the answers of research questions. That method is qualitative.

2.2 Method description

Firstly, I looked into the previous researchers’ report and combined the searches around the online forums, vulnerability database to understand how dangerous the Android root exploits are. With the check app, I got screenshots for vulnerabilities in testing system. Later, I would check the Android formal documents and developers’ blogs to find out whether one crashed app would affect the stability of the system and how SEAndroid affects the rooting process. Here I used command line app to check the running mode of SEAndroid for each devices. After doing the theory research, I came to practice part. I tried to use one-click app in devices or software in PC to root the devices. If this failed, I tried to use custom recovery to get the root access. I would use root checker app to confirm the device’s status and try Google’s API to detect the security level of devices. Finally, I would list the solutions for RQ1, and metrics for RQ2. In analysis part, I would pick up the most suitable solutions according to the results above.

2.3 Reliability and Validity

All my results have the screenshots for references. The test apps in devices or softwares in PC are all latest versions. The test Android systems are the latest version in their version series. Also the testing processes are followed by formal suggestion documents, for example, how to unlock the bootloader in Nexus 10 which I find it in Google’s developer website. However, in the near future, things could change. For example, now there is no rooting method for Android O. Maybe after several beta versions, the developers would figure out how to use the new method to root it.

2.4 Ethical considerations

Before the testing, all the data in devices are formatted. I also used the separate Google account to register the devices and use the Google Play store. The softwares in PC were running in the virtual machine system. Hiit Media company provides me their demo tablet with installed AOSP system. Before the test, I have backed up the system. After doing all the test, I have fully recovered it.
3 Testing Device and System

In this section, I will introduce the testing devices and which Android version they can be installed.

3.1 Samsung Google Nexus 10

Nexus 10 was the first big high-resolution screen tablet developed and marketed by Google, manufactured by Samsung. Hiit Media provides this tablet for me to test. They also used this table as the sample to develop the targeted functions in their future tablet. It is able to use the Android version from 4.2.2 to 5.1.1[3].

3.1.1 Google Nexus 10 Stock Android 4.4.4

Android 4.4 is the first Android System which deploy SEAndroid Enforcing mode for security purpose. But it has only the enforcement on a limited set of crucial domains (installd, netd, vold and zygote)[4]. Android 4.4.4 is the last version of Android 4.4 series. Figure 3.1 shows the detailed information about this version.

![Figure 3.1: Google Nexus 10 Stock Android 4.4.4 with Kernel 3.4.39-gd52af31](image)

3.1.2 Google Nexus 10 Stock Android 5.1.1

From Android 5.0, Google has deployed fully enforcement support to everything (more than 60 domains)[4]. Android 5.1.1 is the last version of Android 5.0 series. Figure 3.2 shows the detailed information about this version.
3.1.3 Company’s pre-configured AOSP system Android 4.4.4

Hiit Media uses pre-configured AOSP system Android 4.4.4 in their devices with the Linux kernel from the Google stock Android 4.4.4. AOSP is Android Open Source Project. It provides information and source code. The developers need to create custom variants of the Android stack, port devices and accessories to the Android platform[5]. Figure 3.3 shows the detailed information about this version.
3.2 Huawei Google Nexus 6p

Nexus 6P is an Android smartphone developed and marketed by Google and manufactured by Huawei[6][7]. It supports the Android system from 6.0.0 to beta O (May be 8.0.0)[8][9].

3.2.1 Google Nexus 6p Stock Android 6.0.1

Google strengthened the security that was first put in place in Android 5.0, essentially making it unfeasible to launch the su daemon with the required permissions just by modifying the /system partition in Android 6.0[10]. Android 6.0.1 is the last version of Android 6.0 series. Figure 3.4 shows the detailed information about this version.

![Figure 3.4: Google Nexus 6p Stock Android 6.0.1 with Kernel 3.10.73-g8bac269](image)

3.2.2 Google Nexus 6p Stock Android 7.1.2

There is no special update related rooting issue from Android 6.0. Android 7.1.2 is the last version of Android 7.0 series. Figure 3.5 shows the detailed information about this version.
3.2.3 Google Nexus 6p Beta Android O

Android O is the next step of the Android System. It brings some new features and security enhancement[11][12]. Figure 3.6 shows the detailed information about this version.
3.3 Samsung S8

Samsung Galaxy S8 is an Android smartphone developed and marketed, manufactured by Samsung[13]. It supports the Android 7.0.0. I will use this device to do some testing expect rooting. Figure 3.7 shows the detailed information about this version.

3.3.1 Samsung Galaxy S8 Stock Android 7.0

Samsung’s self-configured Android 7.0 has the almost latest Linux kernel[14] and additional security method Knox.
Figure 3.7: Samsung Galaxy S8 Stock Android 7.0 with Kernel 4.4.13-11168760
4 SEAndroid

If we want to understand secure Android, we need to investigate the Android Application Sandbox.

4.1 Android Application Sandbox

By investigating the Android runtime mechanism, we could also find out whether single Android application’s crash would affect the stability of the system.

4.1.1 Brief introduction of sandbox model

The sandbox in life is a children’s toy, such as a container filled with balls in KFC. Children can play in it. Also, it plays a role in protecting children which can also be understood as a safe environment.

In recent years, with the increasingly prominent issues of network security, people will use more sandbox technologies in surfing the Internet. From the technical point of view, the original method is to prevent suspicious program access to the system. But the sandbox is to redirect the suspect program to the specified folder when it tries to access the disk, registry or other resources. As a result, the sandbox would eliminate the harm to the system.

For example, GreenBorder builds a secure virtual execution environment for IE and FireFox. Any disk wrote by the user through the browser, will be redirected to a specific temporary folder. In this way, even if the page contains viruses, Trojans, advertising and other malicious programs, they only force to be installed into a temporary folder. It will not harm user’s PC.

For the development of the Java program, the sandbox also has a crucial security significance. Whenever Sandboxes load the code on a remote site and execute locally, security is a critical issue. It can click a link to launch the Java Web Start application. When you visit a web page, all of the Applet will automatically start. If you click on a link or visit a webpage, the user’s machine can install any code; then the criminals may be at this time eavesdropping confidential information, read the financial data or take over the user computer to send ads. To ensure that Java technology will not be used for evil purposes, in the design of Java, SUN company developed a set of sophisticated security model which is security manager. It will check the right to use all the system resources. In the case of default, only those harmless operations are allowed. To allow other operations to be performed, the code needs to be digitally signed, and the user must be digitally certified.

In particular, the procedures in the sandbox have the following limitations:

- You cannot run any local executable program.
- You cannot read any information from the local computer’s file system or write any information to the local computer’s file system.
- You cannot view any information about the local computer except for Java version information and a few harmless operating system details. In particular, the code in the sandbox cannot view the user name, E-mail address and other information.
- The remotely loaded program cannot communicate with any host other than the server where the download program resides, which is called the originating host.
This rule is often referred to as the “remote code can only talk to family”. This rule will ensure that the code will not detect internal network resources of users. In Java SE 6, Java Web Start application can connect to other networks, but it must obtain the consent of the user.

Therefore, in recent years, sandbox technology is very popular, such as 360 browsers, etc.. These browsers have claimed to use the sandbox technology to ensure Internet security. As for the Android system, it is also consciously introduced such a concept.

4.1.2 Sandbox model in Android

In Linux, a user ID identifies a given user; on Android, a user ID identifies an application. The application is assigned the user ID at the time of installation, and the user ID remains unchanged for the duration of the application on the device. Permissions are about allowing or restricting applications (not users) to access device resources.

In essence, Android uses the concept of a sandbox to implement separation and permissions between applications to allow or deny an application’s access to device resources, such as files and directories, networks, sensors, and APIs. To do this, Android uses some Linux utilities (such as process-level security, application-related user and group IDs, and permissions) to enable the application to be allowed to perform.

Conceptually, the sandbox can be shown as Figure 4.8. You can clearly see that both Android apps are in their own basic sandbox or process.

![Figure 4.8: Two applications on different processes (with different user-ids)](image)

Android applications run on their own Linux processes and are assigned a unique user ID. By default, applications running in the basic sandbox process are not assigned permissions, thus preventing such applications from accessing the system or resources. But the Android application can request permissions through the application’s manifest file. For example, Google’s App can recall Google service to login with one-click.

Android applications can allow other applications to access their resources by doing the following:

1. The application declares the appropriate manifest permissions.
2. The application runs with other trusted applications in the same process, thus sharing their data and code access (as shown in Figure 4.9).

![Android application/process space](image)

Figure 4.9: Two applications signed using the same digital signature and assigned the same Linux user-id

Different applications can run in the same process. For this method, you must firstly use the same private key to sign these applications, and then you must use the manifest file to assign them the same Linux user ID, which can use the same value/name to define manifest “android: sharedUserId”.

### 4.2 SEAndroid Introduction

Android’s security model is based on the part of the application sandbox concept; each application is running in their sandbox. In the previous version of Android 4.3, the system creates an independent uid for each application during the application installation and controls the access process based on uid to access the resource. This security model builds on the Linux traditional security model DAC (Discretionary Access Control) to achieve. Starting with Android 4.3, Secure Enhanced Linux (SELinux) is used to define the boundaries of the application sandbox further. As part of the Android security model, Android uses SELinux’s Mandatory access control (MAC) to manage all processes. Even if the process can root (superuser privileges), SELinux can create automatic security policy (sepolicy) to limit privileges process to enhance the security of Android. Starting with Android 4.4, Android has enabled SELinux’s Enforcing mode to work under the default security policy (sepolicy) defined by the AOSP codebase. In Enforcing mode, violations of the SELinux security policy will be blocked; all illegal access will be recorded in the dmesg and logcat. So, by looking at dmesg or logcat, we can collect information about breaches of the SELinux policy to improve our software and SELinux strategy.

Manufacturers can define different SEAndroid policy for their devices according to different needs. The policies on a Samsung device may be significantly different from the policies on a Google Nexus device. [15]
4.3 How to root in SEAndroid

Material Terminal

Material Terminal developed by Yaroslav Shevchuk makes you be able to access your Android’s built-in Linux command line shell[16]. I will use this app to check SEAndroid running mode. From figure 4.10, it shows that enforcing mode is activated on all test devices from Android 4.4.4, Android 5.1.1, Android 6.0.1, Android 7.1.2, Android O, Samsung S8 Android 7.0.

Figure 4.10: Enforcing mode is activated on all test devices

My test devices are all in SEAndroid Enforcing mode. SEAndroid will deploy to all the future Android phones. I would focus on the root mechanism for SEAndroid. If you want to learn about old root methods before Android 4.3, you can read this article [2] which also provides me ideas of my research. Figure 4.11 describes how a original app to gain the root access from su binary on Android versions 4.3 and later. The dashed boxes mean process boundaries. Security contexts (domain) have underlined labels. [2].

Figure 4.11: The process to gain the root access from su binary on Android versions 4.3 and later

In SEAndroid, a security context called a domain marks each process in the system. When a process attempts to perform an operation on a resource called an object in a SEAndroid term, the object manager in the kernel intercepts the requested operation and checks the request based on the centrally managed MAC policy to determine whether the request should be allowed or not. Therefore, in enforcing mode of SEAndroid, execut-
ing the root UID implementation process does not change its security context, and the SEAndroid MAC policy still limits such a process.

Although you can flash some replaced files into the boot or system partitions to lower these security controls, such a radical approach is not the root permission manager app’s developer Chainfire wants. He said in his website “Note that various custom kernels and firmwares switch SELinux back to permissive for Android 4.4 and newer. This completely disables all the new security features SELinux brings, rather than relaxing only the areas we absolutely need to get our apps to function.”

Chainfire[15] has brought an idea to root the device under the enforcing mode. Firstly, we need to understand three important security contexts in SEAndroid root process:

- **daemonsu** su binary is known as “switch user”, “super user”, or “substitute user”. It is the middleman between apps and system when the application requests root access. It is the key for the whole rooting process. In SEAndroid, this middleman divides into process su and a su daemon (denoted as daemonsu). The su binary will forward all the requests received from apps to daemonsu. daemonsu will run as a daemon process with root uid and init security context. daemonsu is forked from init. It can inherit the domain of init.

- **init** Android’s first process is init. All the other processes are forked from init directly or indirectly. In Google Android source help website, it states “No processes other than init should run in the init domain”.

- **zygote** In Android, all the application processes and SystemServer process are forked by zygote process. zygote has been in the enforcing mode since Android 4.4.

Then, the whole rooting process is beginning:

1. Chainfire provides two approaches to modify the system to let the daemonsu be invoked by init[15]. One approach is to change the system partition. Another one is to modify the boot image. “From Android 6.0, Google strengthened the security that was first put in place in Android 5.0, essentially making it unfeasible to launch the su daemon with the required permissions just by modifying the /system partition.” stated by howtogeek.com[10]. The detail would be discussed in later section. Also, daemonsu patches the MAC policy during boot time to un-confine the init domain and to allow su to connect to the local socket created by the daemonsu.

2. At boot time, the first process init sets its domain as init. When init starts the zygote process, the security context of the zygote is automatically marked. Also, init forks daemonsu which labeled the security context of the init and the root uid.

3. All Android applications are dispatched by the zygote process, and zygote assigns the application to an untrusted app domain based on the UID during the installation and a system configuration file. Note that when su is invoked by the application, su inherits the application’s domain which is untrusted app.

4. su then connects to daemonsu’s local socket. Upon receipt of the request, daemonsu will restart a new process and then execute the input command under the root uid and init security contexts. Then this process returns the output to the original app.
5 Taxonomy of Android Rooting Methods

We need to modify the system partition or boot image to achieve the root purpose. Here we have two approaches. One approach is the post-boot method that is to gain a temporary root access and use this root access to configure the system partition. The other one is the pre-boot method that is to flash self-configured recovery or system image to replace the system partition or boot image. Figure 5.12 tells the different requirements and processes between pre-boot and post-boot methods[2].

The former researchers has described the taxonomy of Android Rooting Methods as Fastboot/Download mode, Custom recovery, Bootable SD Card, Rooting apps or tools and Privileged ADB[2]. But here, I will introduce the taxonomy as system root and systemless root according to Chainfire and other developers’ latest rooting methods. It also would combine the taxonomy above. And there is another reason I do this. System root applies the Android version before 6 and systemless root applies the Android version after 6[10].

5.1 System root

Just as the name, the method is to modify the /system partition to achieve root access. The simplest way is using the rooting apps or tools. They use Android root exploits to achieve this purpose.

5.1.1 Android Root Exploit Impact and Coverage

There are four categories based on the Android Architecture described by the former researchers, Linux Kernel, Vendor-Specific Kernel or Drivers, Libraries Layer, Application and Application Framework. [1]. Figure 5.13 shows the Android Software Stack.
Figure 5.13: Android Software Stack[17]
Table 5.1 shows the different Linux kernel versions in different AOSPs. Linux kernel is the foundation of the Android platform. Linux kernel can provide key security features to the Android and allows manufacturers to develop hardware drivers for their own devices. The newer version of Linux kernel could optimize the code of old ones. Also, latest version fixes the bugs or vulnerabilities and brings new hardware functions. It is not easy to find a root exploit in a Linux kernel. But if you find one, it will affect all the devices which use this Linux kernel. For example, CVE-2014-3153 touched all the Linux kernel through 3.14.5. (CVE: Common Vulnerabilities and Exposures). Developer geohot used this vulnerability to develop an app called TowelRoot to gain root access to the Android devices. According to his website[15], it should support all Android phones’ system version before June 3, 2014. Most of the popular one-click software also used this vulnerability. But this did not mean that all the Linux kernel versions below 3.14.5 were not safe anymore. The manufacturers can directly deploy an update patch to their old kernel.

Back to Android, it cannot get these kernel vulnerabilities patched in a timely manner like iOS. According to the report from Baidu X-Lab[19], the researchers gave three causes:

1. The long patching chain delays the patch effective date;
2. Fragmentation makes it challenging to adapt the patches to all devices;
3. Capability mismatching between device vendors and security vendors.

<table>
<thead>
<tr>
<th>Android Version</th>
<th>API Level</th>
<th>Linux Kernel in AOSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5 Cupcake</td>
<td>3</td>
<td>2.6.27</td>
</tr>
<tr>
<td>1.6 Donut</td>
<td>4</td>
<td>2.6.29</td>
</tr>
<tr>
<td>2.0/1 Eclair</td>
<td>5-7</td>
<td>2.6.29</td>
</tr>
<tr>
<td>2.2.x Froyo</td>
<td>8</td>
<td>2.6.32</td>
</tr>
<tr>
<td>2.3.x Gingerbread</td>
<td>9, 10</td>
<td>2.6.35</td>
</tr>
<tr>
<td>3.x.x Honeycomb</td>
<td>11-13</td>
<td>2.6.36</td>
</tr>
<tr>
<td>4.0.x Ice Cream San</td>
<td>14, 15</td>
<td>3.0.1</td>
</tr>
<tr>
<td>4.1.x Jelly Bean</td>
<td>16</td>
<td>3.0.31</td>
</tr>
<tr>
<td>4.2.x Jelly Bean</td>
<td>17</td>
<td>3.4.0</td>
</tr>
<tr>
<td>4.3 Jelly Bean</td>
<td>18</td>
<td>3.4.39</td>
</tr>
<tr>
<td>4.4 Kit Kat</td>
<td>19, 20</td>
<td>3.10</td>
</tr>
<tr>
<td>5.x Lollipop</td>
<td>21, 22</td>
<td>3.16.1</td>
</tr>
<tr>
<td>6.0 Marshmallow</td>
<td>23</td>
<td>3.18.10</td>
</tr>
<tr>
<td>7.0 Nougat</td>
<td>24</td>
<td>4.4.1</td>
</tr>
<tr>
<td>7.1 Nougat</td>
<td>25</td>
<td>4.4.1 (To be updated)</td>
</tr>
</tbody>
</table>

Table 5.1: Android Version And Kernel[18]
Figure 5.14: Days from advisory publication date to July 2016 for the three famous “one-to-root-all” kernel vulnerabilities

Figure 5.15: Vulnerability statistics collected from Chinese Android device (July 2016)

From Figure 5.14 and Figure 5.15, we can see that so many users were still in danger of root exploit attack even after months or years of the vulnerability’s disclosure. The root exploits were discovered but their devices could not get the updated patches in time.

**Vendor-Specific Kernel or Drivers**
The manufacturers need to develop their hardware drivers in the customized Linux kernel according to their needs. And they need to provide security patches for their own devices. There is also an advantage about this. If the manufacturers did patch their devices in time after finding root Linux kernel exploits, then the devices could be secure from some “one-click” root software. The root software’s database needs to add different root exploits for various vendors. Google is not a vendor who likes to keep its devices to use the latest kernel. Even for the last year’s flagship Google Pixel XL, it only has the Linux Kernel 3.18. Comparing to Google, Samsung’s this year’s flagship Galaxy S8 has jumped to 4.4.13. The latest stable Linux kernel is 4.11.1.

**Libraries Layer**
Libraries Layer exploits target the Android libraries or external libraries. They could have a significant impact since multiple applications may use the library. For example, ZergRush exploit (CVE-2011-3874) would affect Android 2.2 - 2.3.6. In that time, almost
every root tools will include it or implement from the open source zergRush.c. Lately, CVE-2014-7911 found by Jann Horn used a bug in ObjectInputStream which could lead to run any code on the operating system. This vulnerability affects the Android version before 5.0.

**Application and Application Framework**
This layer exploits are usually found in setuid utilities, system applications, or services. If you are lucky, you could find one by accident. Because this kind of exploits almost seems to be the back doors which are left by the manufacturers, such as XoomFE bug and CVE-2012-2949. However, this kind of exploits only indicated a limited impact.

**Publicly known vulnerabilities on test devices**

**Vulnerabilities check app**
The Android app called *X-Ray*

“Back in 2012, the Duo Labs security research team released the first-ever vulnerability scanner for Android-based devices known as X-Ray. X-Ray is an app anyone can download that safely scans for vulnerabilities on your Android phone or tablet, allowing you to assess your current mobile security risk.” according to the Duo website[20].

From VTS’s github page, we can know that X-Ray tries to detect the vulnerabilities by actually attempting the exploit[21]. But now both X-Ray and VTS share a same vulnerability database[22]. I will use this app to check the vulnerabilities for different version of Android.

The Android app called *VTS for Android*

Android Vulnerability Test Suite is an open source project released by NowSecure in 2015[21]. “This implementation takes care to not include checks that could cause instability problems for the end user and therefore may omit checks that could cause these types of issues.” according to the VTS’s github page[21]. I will use this app to double check the vulnerabilities for different version of Android.
**Result**

**Google Nexus 10 Stock Android 4.4.4**

*Figure 5.16* shows 13 vulnerabilities including well-known kernel exploits TowelRoot (CVE-2014-3153) and Ping Pong Root (CVE-2015-3636) in Google Nexus 10 Stock Android 4.4.4. This version was much vulnerable to root.

![Figure 5.16: The result from VTS and X-Ray in Google Nexus 10 Stock Android 4.4.4](image)

**Google Nexus 10 Stock Android 5.1.1**

*Figure 5.17* shows 0 vulnerability in Google Nexus 10 Stock Android 5.1.1. This version was installed with Android security patch 2016-04-01 which actually fixed all the well-known vulnerabilities.

![Figure 5.17: The result from VTS and X-Ray in Google Nexus 10 Stock Android 5.1.1](image)

**Google Nexus 6p Stock Android 6.0.1**

*Figure 5.18* shows 1 vulnerability in Google Nexus 6p Stock Android 6.0.1. It was error in VTS but it was considered as “Vulnerable” in X-Ray. It may be a false alarm. Still we can see that Google has fixed almost all publicly known vulnerabilities in newer system.
Figure 5.18: The result from VTS and X-Ray in Google Nexus 6p Stock Android 6.0.1

Samsung Galaxy S8 Stock Android 7.0

Figure 5.19 shows 3 vulnerabilities in Samsung Galaxy S8 Stock Android 7.0. It was unbelievable that S8 still had some old vulnerabilities, since it has the almost latest Android version and Linux kernel. Also, VTS may have some compatibility issues. Sometimes it showed the device was vulnerable to CVE-2013-6282. But after several re-trys, it may show the device was not vulnerable at all. However other vulnerabilities in X-Ray were stated errors in VTS. They may be a false alarm.
Figure 5.19: The result from VTS and X-Ray in Samsung Galaxy S8 Stock Android 7.0

**Google Nexus 6p Stock Android 7.1.2**
Both VTS and X-Ray cannot run tests in this system. Every time I clicked scan the vulnerabilities, the apps would crash out.

**Company’s pre-configured AOSP system Android 4.4.4**
Figure 5.20 shows 13 vulnerabilities including well-known kernel exploits TowelRoot (CVE-2014-3153) and Ping Pong Root (CVE-2015-3636) in company’s pre-configured AOSP system Android 4.4.4. This version was much vulnerable to root. It is similar to the Google’s stock Android 4.4.4.

Figure 5.20: The result from VTS and X-Ray in Company’s pre-configured AOSP system Android 4.4.4

**Google Nexus 6p Beta Android O**
Both VTS and X-Ray cannot run tests in this system. Every time I clicked scan the vulnerabilities, the apps would crash out.
5.1.2 Methods

Rooting apps or tools
Before Android 6, there are some system or kernel vulnerabilities which can be exploited to gain temporary root access. The one-click root apps or tools use this root access to configure the system partition and install `su`.

We can also use the custom recovery to install system root. But this requires unlocked bootloader.

Unlock bootloader
The manufacturer always lock the device’s bootloader by default for the safety of device. Google’s devices usually give the detail unlock tutorial in their website, since the Google Nexus series or Pixel now are welcomed by developer for developing purpose. But for new Samsung devices, unlock the bootloader would trigger one “button” in hardware. After booting, the system would check this value in hardware. This may cause you lose the warranty and all the Knox functions. Also when you unlock the bootloader, you will trigger the factory reset which would reformat the `userdata` and `cache` partitions for avoiding malicious OS image to access existing user data. Different devices from other manufacturers would have other approaches to unlock the bootloader.

Custom recovery
After unlocking the bootloader, we are able to use ADB (Android Debug Bridge) tool for common devices or ODIN exclusive for Samsung devices to install custom recovery.

A Linux kernel, a RAM disk with various low-level executables and configuration files, and a recovery program, form a minimal Linux OS called Recovery OS. Manufacturer’s over-the-air (OTA) update packages deliver to the user and use a script in it to apply post-ship system updates in stock recovery OS. OTA update packages would be signed by the device manufacturer. Stock recovery OS would typically check it and decide whether to deploy the updates. All existing emphasu distributions provide installation packages in a form of OTA update packages but without manufacturer’s signature. Then, we need to flash into the third-party recovery for installing the self-configured packages, including `superSU` for root access. Since the recovery OS is stored in a partition similar to `system` and `data` partitions, we are able to replace it with a custom recovery OS, once a temporary root access has been obtained or use ADB tool to flash.

For now, Team Win Recovery Project is the only latest updating custom recovery. It has the official TWRP APP in Google Play store. But the app requires root permission for flashing its custom recovery. So in other way, we need to go to their official website, search about our devices and then download the recovery image. Next step is to connect the phone to the local pc and use adb tool to flash into the TWRP recovery image with unlocked bootloader. Then, we can enter this custom recovery to flash the `su` package without considering manufacturer’s signature. An OTA update package can add or alter system files instead of replacing the whole system image. `su` also can coexist with the existing stock system on the device. The OTA update’s updater script is usually written by a special `edify` script language. However, from the former researchers, they found that “all su OTA packages use traditional UNIX shell, so that the same installation script can be reused by other rooting methods (e.g., bootable SD card, privileged ADB, one-click rooting apps).”[2] Team Win Recovery Project is the only updating popular custom recovery in Android. It is also an open source, community project[23][24]. Figure 5.21 shows the differences between stock recovery and third party TWRP recovery.
Figure 5.21: The top picture is the Nexus 6P’s stock recovery. The bottom picture is the TWRP recovery for Nexus 6P
"/system" method

Figure 5.22 shows the process when SuperSU installs the system root. We can clearly see that the installer mounted /system and modified it.

![Image](image.png)

**Figure 5.22**: The process when doing /system rooting

5.2 Systemless root

A Linux kernel and a RAM disk form a Android boot image. The RAM disk is a small partition image. This image contains a root filesystem such as rootfs. This filesystem is read-only by the kernel at boot time. The first Linux process init would be started to mount the rest of file system once rootfs is mounted. Also init would perform initialization procedures based on the configuration files. We can create a boot image contains customized commands and scripts. As a result, system would execute our desirable command, including root access. Chainfire’s systemless root is based on this idea. This root method applies the device with Android 6.0 or later version. And it requires the unlocked bootloader and custom recovery.

Systemless Root

Figure 5.23 shows the process when SuperSU installs the systemless root. Installer firstly created the image and mount it. This is using the loopback device feature. Then when the system boots up, it will use the bind mount feature to replace the original path to our new created path. This action achieves the need that executing different archives without modifying the system partition.[25]
Figure 5.23: The process when doing Systemless rooting
6 Penetration tests for different Android Versions

6.1 Root Checker

“Root Checker is a free, fast and easy to use one-click root checking app to verify if your device has proper root access or not (superuser or su).” according to the description of the app in Google Play store[26]. I will use this tool to check whether the device is rooted successfully or not.

6.2 One-click root software

I will choose the XDA forum’s recommended one-click root softwares in 2016 to test[27]. They are KingRoot, OneClickRoot, KingoRoot and CF-AutoRoot.

After reading the description document about CF-AutoRoot, I find that this method is mainly for Samsung device[28]. “A modified recovery and cache partition will be flashed, which will install (only) SuperSU, then clean-up the cache partition and re-flash the stock recovery.” Although it seems one-click when using ODIN, it just transfers the manually flashing su in the custom recovery to the process of automation. As a result, I would pass CF-AutoRoot.

When I tried to use OneClickRoot, I found it was a paid software. You need to pay 39.95 dollars to get the root function. The experts would think this price is unbelievable. The normal customers would think it is expensive to root the device and they may give up trying to do so. As a result, I would also pass OneClickRoot.

KingRoot
“KingRoot No.1 Root Android Phone In One Click”. It has an Android app and an windows host client[29].

KingoRoot
“KingoRoot - The Best One Click Android Root Apk/Software for Free”. It also has an Android app and an windows host client[30].

TowelRoot
This app is based on kernel root exploit CVE-2014-3153[31]. And developer geohot implemented a app to use this exploit to root the device[32]. This will be the additional test for company’s AOSP device.

6.3 su binaries

SuperSU
“SuperSU is one of the world’s most popular tool for root apps. SuperSU allows advanced management of Superuser access rights for all the apps on your device that need root.” according to the official SuperSU website[33][34]. The author Chainfire created the systemless root which allows Android 6.0 and 7.0 to be rooted. phh’s SuperUser

phh’s SuperUser uses the same idea of systemless root. It is an open source project[35][36][37][38]. Since Chainfire sells the SuperSU to a company called Coding Code Mobile Technology, the community is afraid of backdoor trick in future SuperSU. Then phh’s SuperUser came out which used the new idea with old out of update original CM superuser app.

Magisk
Magisk is a Root and Universal Systemless Interface[39][40]. Magisk is also an open source project. It contains a function called MagiskSU which based on phh’s Superuser. It has two main interesting feature “1. Anything that previously modifies /system can be loaded with Magisk. I ported Xposed and ViPER4Android just as an example. 2.
Use apps/features that uses Google’s SafetyNet detection” according to their xda forum page[39].

6.4 Result

The one-click apps or tools use these vulnerabilities to configure the system partition and install su. I will try KingRoot and KingoRoot one by one. This means that if the KingRoot is failed to root, then I will try KingoRoot. If all these three are failed, I will try to use manual custom recovery root. About the manual custom recovery root, I would try SuperSU firstly, then phh’s SuperUser and Magisk. The details of these three su binary would be explained in the later sections.

Google Nexus 10 Stock Android 4.4.4

Figure 6.24 shows the process. The first screenshot shows that when you install the King-root app downloaded from its website, the system would warn you that this app is dangerous. I believe this warning is from the google service package which has been updated automatically in the device. The UI is obvious from the latest Android design. Then in the next screenshot, we can see “Processing(1/13)”. I think there are thirteen approaches for rooting this device with this Android version. From the above section, I have described this Android version 4.4 has been found many vulnerabilities including two critical kernel exploits. It is not surprising that the device is been rooted successfully by KingRoot one-click app. In the end, KingRoot app has contained its own root manager function. SU binary was found in system partition /system/bin/su.

![Image of Google Nexus 10 Stock Android 4.4.4 rooting process](image)

Figure 6.24: Google Nexus 10 Stock Android 4.4.4 rooting process

Google Nexus 10 Stock Android 5.1.1

Figure 6.25 shows the process. With a wrong click, I tried the KingoRoot firstly. It was surprising that even Google has fixed all the known vulnerabilities here with latest security patches, it still got rooted by one-click root app. I also tried the device with locked bootloader. It still showed the same result. SU binary was found in system partition /system/bin/su.
Google Nexus 6p Stock Android 6.0.1

Figure 6.26 shows the first try to root Android 6.0.1 with apps. From the first screenshot, we see the similar warning in the section above. For this version, KingRoot only has four approaches to root the device. Still, it failed in both app or pc side. Then I tried KingoRoot, it showed the same result. The device has included the latest Android security patch for Android 6.0.1. Also we can see Google’s effort to prevent root is successful for one-click root after Android 6.
Figure 6.26: Google Nexus 6p Stock Android 6.0.1 rooting process of apps

Figure 6.27 shows the second try to root Android 6.0.1 with custom recovery. I downloaded the official TWRP recovery for Nexus 6p in host PC firstly. Later, I downloaded the latest version of superSU on the device. After that, I opened the device’s USB debugging mode and connected to the host PC. From PC command terminal, I used adb command *adb reboot bootloader* to let the device enter the bootloader. This step is not necessary, since we can turn off the device and use Power button plus Volume Down button to enter bootloader. Then in host pc’s terminal, I typed in *fastboot flash recovery twrp.img* to flash recovery image. Here, if we reboot device directly, the stock recovery would be flashed back automatically. We need to choose the recovery option at once. We can keep the system partition read-only, since the superSU root could be systemless and do nothing to the system partition (I have tested both options that they were all working). Then, in the TWRP recovery, I choose to flash in the superSU package I prepared before. After flashing, I can check the root was successful and working properly. SU binary was not found in system partition but in here /su/bin/su. This proved that it was a systemless root.
Google Nexus 6p Stock Android 6.0.1 rooting process of custom recovery

With the similar process above in Android 6.0.1, Android 7.1.2 can be rooted successfully by doing the flash in the custom recovery. SU binary was not found in system partition but in here `/su/bin/su`. This proved that it was a systemless root. Figure 6.28 shows the successful rooting result.

Figure 6.28: Google Nexus 6p Stock Android 7.1.2 rooting result through custom recovery

---

32
Company’s pre-configured AOSP system Android 4.4.4

Even with the similar vulnerabilities in Android 4.4.4, pre-configured AOSP made all the one-click apps failed to root, such as KingRoot, KingoRoot or TowelRoot (This one mainly targets to the CVE-2014-3153 vulnerability which the device had.). The reason is that in their app’s database, there is no device system record for our pre-configured AOSP system. Then I used manual flash SuperSU in the custom recovery to root the device like Android 6 and 7. SU binary was found in system partition /system/bin/su. This is a system root. Figure 6.29 shows the successful rooting result.

![Figure 6.29: Company’s pre-configured AOSP system Android 4.4.4 rooting result](image)

Google Nexus 6p Stock Android O

All three su binaries are failed to root. It was not surprising, since Google could tight the security level step by step against the rooting. Figure 6.30 shows the failed rooting result.

![Figure 6.30: Google Nexus 6p Stock Android O rooting result](image)
7 Samsung Knox

“Knox is Samsung’s defense-grade security platform built into our latest mobile devices.” according to the Knox website[41]. As a normal customer, you can store your secret apps or information in Knox container. That is another hardware level virtual machine. You can install a app outside the container and also inside the container. But they are not sharing one database. You can open them at same time, or you can just simple delete one but the other one is not affected at all. When you enter the container, system requires you to provide Iris, fingerprint or passcode check.

Also Knox has a special feature called Knox Warranty Bit. If it detects non-Knox kernel or unlocked bootloader, it would be burned to 0X1 from 0X0. It is a hardware level change. You are no longer able to use the Knox container again. The data which is encrypted and stored in Knox container can no longer be retrieved. And it may affect the warranty about the device. The only way to recovery the Knox function is to replace the hardware parts.

Then, it is dangerous to root the latest Samsung devices. As we knew, in Android 6 and 7, we must unlock the bootloader to install custom recovery and install su binary to root the device. With the setting like Knox, we need to consider the result of rooting.
8 SafetyNet API

SafetyNet Test

“SafetyNet Test verifies the device with SafetyNet compatibility tests. SafetyNet api is a part of Google Play Services Framework. Some apps use this api to check whether the device has been rooted or tampered.” according to the description of the app in Google Play store[42]. I will use this tool to check whether the system would pass the Google SafetyNet api test.

The SafetyNet API is Google’s official offer to check if the device is available through Android compatibility testing. It will use the hardware and software features of the device to generate profiles for compatibility check. This API is not designed to check whether the device is root, even if not root, just use a third party ROM, or use some of Chinese low-cost white card phone, can not be verified. If the phone cannot pass through its verification, it cannot use the official Android Pay for shopping. SafetyNet is supported from Google Play Service 2.3 and above[43].

The whole process of SafetyNet is:

1. Adding an API Key to the Manifest.
2. Obtain a single use token.
3. Send the compatibility check request.
4. Read the compatibility check response.
5. Verify the compatibility check response

Figure 8.31: Android 4.4.4 and Android 5.1.1 with unlocked bootloader BEFORE rooting
Figure 8.32: Company’s pre-configured AOSP system Android 4.4.4 with Unlocked bootloader BEFORE rooting

Figure 8.33: Android 4.4.4 and Android 5.1.1 with unlocked bootloader AFTER rooting
Figure 8.34: Android 6.0.1, Android 7.1.2 and Android O with Unlocked bootloader BEFORE rooting

Figure 8.35: Android 6.0.1, Android 7.1.2, Android O and Samsung Galaxy S8 Android 7.0 with Locked bootloader BEFORE rooting

Figure 8.36: Android 6.0.1, Android 7.1.2 and Android O with Locked bootloader AFTER rooting
9 Result

9.1 Android Root Exploit Impact and Coverage

The Android Root Exploit danger level from highest to lowest would be

1. the kernel exploits,
2. the exploits targeting libraries that are used by Android system processes,
3. exploits targeting system applications or services,
4. exploits against vendor-specific device drivers, applications, and programs

Android 4.4.4 and Company’s self-configured AOSP 4.4.4 have much more vulnerabilities than others. They are supposed to be rooted easily by using such a lot of root exploits. Comparing to Android 4.4.4, Android 5.1.1 has a new kernel with small version change. But it still fixed all the Linux kernel root exploits found in 4.4.4. From Android 6.0.1 and later version, Google has fixed all the well-known vulnerabilities. It existed some bugs when using the app to detect the vulnerabilities.

9.2 Android Application Sandbox

In normal cases, the application runs in a sandbox and when it crashes, it will not affect the stability of the system. In special cases, the application could use the root exploits to jump out of the sandbox and root the devices, such as Towelroot and other one-click root apps.

9.3 SEAndroid

SEAndroid has increased security control in Android system since Android 4.4 with enforcing mode. But it is still possible to get rooted with new logic idea. Before Android 6.0, you are able to modify the system partition to gain root access. However, after Android 6.0, Google made it unfeasible to launch the su daemon with the required permissions just by modifying the /system partition. You have to use systemless root to modify the boot image and gain root after booting.

9.4 Taxonomy of Android Rooting Methods

We have multiple choices to flash the su binary to the devices. When we use Android 5.1.1 and the version before, we can simply use one-click rooting apps to install su binary. But after version 6, we should use custom recovery to flash in the su binary which require unlocked bootloader. Here, I also monitored the two kinds of su binary installation in the custom recovery. We can see the obvious differences between system and systemless rooting.

9.5 Penetration tests for different Android Versions

Android 4.4.4 and Android 5.1.1 are easier to get rooted among other system. Both of them can be rooted with one-click root apps with locked bootloaders. Android 6.0.1 and 7.1.2 need the unlocked bootloader to install custom recovery. We can flash in su binary through the custom recovery. In addition, for company’s AOSP Android 4.4.4, I have to
used the same methods above to flash the su binary. One-click root would not recognize the device. In Android O, Google seems to tighten the security control again. For now, there is no way to root the device with Android O.

9.6 Samsung Knox

Samsung Knox shows the Samsung’s determination to their device’s security. Especially, you cannot get rooted without unlocking the bootloader since Android 6. If you want to do the rooting things, you have to give up the Samsung’s safety service and even warranty of your device. This is a strict move.

9.7 SafetyNet API

SafetyNet API would detect the device’s status. Unlocked bootloader with unrooted Android 4.4.4 and 5.1.1 would pass the test. If you root these two versions Android system, you would not pass the test with result “fail Basic Integrity and fail CTS profile match”. Unlocked bootloader with unrooted Android 6.0.1, 7.1.2 and O would not pass the test with result “fail CTS profile match”. If you root the device, the result would get failed response signature validation. If you want to pass the test, you have to get a locked bootloader and unrooted system in Android 6.0.1, 7.1.2 and O. As for the company’s AOSP 4.4.4, the test would fail with or without rooting. Samsung’s device galaxy S8 can also pass the test with stock system.

9.8 Answers to the RQ1 and RQ2

What solutions exist to prevent rooting in Android System?

- Android Version
  1. AOSP Android 4.4.4 with patched kernel or blocked settings
  2. AOSP Android 5.1.1 with patched kernel or blocked settings
  3. AOSP Android 6.0.1, Android 7.1.2 or Android O
  4. AOSP Android O(go)

- Detection methods
  1. Hardware Detection like Samsung Knox
  2. Google SafetyNet API

What parameters exist to measure the solutions above?

1. Cost. The company has limited budget to build a ad-safe tablet.

2. Availability. Google only provides the formal support for Android 4.4.4 and Android 5.1.1 in company’s demo tablet Nexus 10.

3. Customer Experience. Customer may want to install the app from other app store or Internet instead of Google Play Store. And also customer wishes to be able to transfer the data between PC and their tablet. In addition, the device should not be too laggy or old bad-looking UI.
10 Analysis and Suggestions to Company

Before Android 6, you are able to find the root exploits and achieve the rooting function by modifying the system partition, such as one-click root apps or flash-in su binary. But after Android 6, even if you have found some root exploits, it is still not possible to root the device by modifying the system partition anymore. su binary work mechanism has changed. You have to patch the boot image during the booting. If you want to patch the boot image, you have to install the script and extra partition in custom recovery. Then if you want to install the custom recovery, you have to unlock your bootloader firstly. If the company wants more strict control measure against the rooting, they can deploy the hardware lock like Samsung Knox. Since the customers are not owning the device, they are not allowed to root the device. The hardware lock can make the device unusable and send the repair request to the support team. In addition, Samsung has full production control and strong budget to achieve Knox function which is also approved by the U.S. Department of Defense. The hardware detection methods like Samsung Knox are too expensive for the company.

AOSP Android 4.4.4 with patched kernel or blocked settings
It is possible to fix the kernel exploits by modifying the kernel content. For example, in theory, TowelRoot should affect all the released Linux kernel through version 3.14.5. But Google has fixed the problem in Nexus 10’s kernel 3.4.67. Also, Nexus 6p’s kernel is 3.10.73 which is also not affected by this root exploit. If the company insists to use the existing AOSP 4.4.4, they needs to fix the well-known kernel and system level vulnerabilities. Although one-click apps is not working on this self-configured system now, they may support it in the future when the devices are deployed widely. We are able to block customer to install third-party app not from Google Play Store which does not have any root tools. But it will affect the user experience. In addition, some other malicious apps or scripts would try to trigger the existing vulnerabilities and do something terrible. Since our device may store the family’s members’ health status, it is important to keep the data of confidentiality and Integrity. However, fixing all these vulnerabilities would cost a lot of time and human resource.

AOSP Android 6.0.1, Android 7.1.2 or Android O
The next idea for company is to use the latest AOSP 6.0.1 or 7.1.2. They have fixed all the well-known vulnerabilities. Google also provides the security level patch every months. These methods keep the device secure enough. Both of them have cut off the possibility of rooting by one-click root apps. Modern UI design is also a bonus for these two versions. The company needs to increase the control of bootloader. The latest methods for rooting 6.0.1 and 7.1.2 must unlock the bootloader. And for our testing device, it also required the USB-debugging mode to enter the bootloader mode. We need also increase the control for USB-debugging mode from device’s developer options. We can set up that only the support team is able to active these two functions. In addition, The company needs to register their system with Google to get verified in SafetyNet API. When the device cannot pass the test after booting, it would notify the support team who will deal with the rest. Also, the company needs to modify the existing kernel version (maybe from Stock Android 5.1.1) to be compatible with latest versions. However, there are still some disadvantages in this solution. If the company’s final product will have the similar hardware as Nexus 10 which is almost 5 years old, the clients may feel slow and laggy when using latest Android versions.
RQ3: Which technology is the most suitable solution for the company?

*AOSP Android 5.1.1 with patched kernel or blocked settings and SafetyNet API for detection of root*

The better idea is to use AOSP 5.1.1 and later updated to Android O(go). As my test result showed, stock Android 5.1.1 has fixed all the well-known vulnerabilities. Google also provides latest security level patch in 2016-04-01 comparing to Android 4.4.4. Also it is easy to pull the kernel off from stock Android 5.1.1 to the AOSP 5.1.1. This would decrease the development costs. Modern UI design is also a bonus. In addition, Android 5.1.1 is usable in Nexus 10 which will not cause too much lag. The company also needs to do the same things as I described before, increasing control of bootloader and developer options, verify in SafetyNet API. The problem is that one-click root apps are able to root the device with Android 5.1.1. But with AOSP 5.1.1, the apps may be blind for a while and we can add blacklist to the system for blocking installing of these apps.

*AOSP Android O(go)*

Google has announced the Android O and a light version Android Go. Android Go is developed for inexpensive devices with limited hardware. According to my result, Android O has not been rooted with Effective technology until now. Android Go would inherit the security feature from Android O. But Android Go will come out next year, it will take time to wait for it.
11 Conclusion

Google has increased control against the rooting step by step. They brought the SEAndroid developed from SELinux. This move disabled all the rooting method before. However, the brilliant developers still find the way to modify the system partition to install the su binary. Also, they made use of root exploits to achieve one-click root. Later, Google increased the security level again that you cannot modify the system partition to require root permission. Then the developers thought of a systemless rooting method by creating a new partition with scripts and modifying the boot image from booting, to achieve rooting. Until now, Google introduces the new Android O which has disabled the systemless rooting method. It is a long battle between rooting and preventing rooting. The developers try the bottom line of Android again and again. Google has to increase the security level version by version. Rooting is also a significant security issue in Android system. If you wish to have a secure device, you must take care of the root permission. Finding the solution for avoiding rooting is the key for the system integrity. Android is the most popular mobile system in the world. The security of Android is related to more than 1.4 billion users’ privacy.

In the end, the paper provide some solutions for Hiit Media’s need of their free-distributed tablet according to my test findings. The solutions could also apply to other companies who have similar purposes.

11.1 Future research

Android system has a feature which is fragmentation. Different manufacturers almost have various Linux kernel and Android versions. The fragmentation can be an advantage that manufacturers can modify the devices according their special functions with specific hardware. That is why the Android environment is colourful and diverse. There should be more devices for testing to support my paper. In addition, there is no rooting method for Android O until now. It is interesting to find out what kind of “spear” the outstanding developer would use to penetrate the “shield” of Google.
References


