**Lab Android Device Rooting Attack**

**By Marco Lin**

**Task1: Build a simple OTA package**

Step 1: Write the update script

Explanation: in this step, we created two script: update-binary and dummy.sh under the META-INF/com/google/android folder:

一張含有 室外 的圖片

自動產生的描述

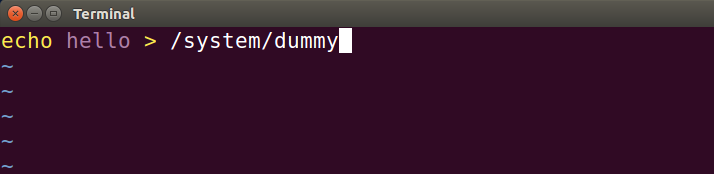
1. update-binary: make it executable.

一張含有 螢幕擷取畫面 的圖片

自動產生的描述



1. dummy.sh



Step 2: Build the OTA Package

Explanation: create a zip file of the entire package

一張含有 文字 的圖片

自動產生的描述

Step 3: Run the OTA Package

1. access the recovery OS of android, and check the IP address of recovery OS

一張含有 螢幕擷取畫面, 坐, 文字 的圖片

自動產生的描述

1. send the zip file from seedUbuntu VM to android recovery OS, and place it into the /tmp folder of the recovery OS.

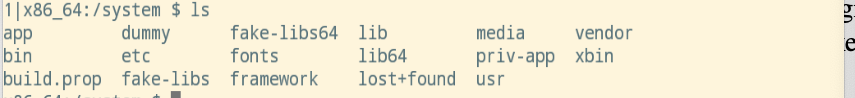
一張含有 室外, 文字 的圖片

自動產生的描述

1. unzip the file on recovery OS and run the update-binary一張含有 螢幕擷取畫面 的圖片

   自動產生的描述
2. go back to Android VM, and see the contents of /system folder and find that out attack is successful with dummy being created in the folder.

Explanation: As we can see, the file is being created on the Android VM



Explanation: We create the OTA package and export the OTA package to the recovery OS. The update-binary file does automatically whatever we are supposed to do so that the attack is successful. The update-binary file first copies the dummy file from the unzipped folder to the system/xbin folder. It then gives executable permission to the dummy file. We then place a line of code in the init folder such that the dummy file is executed when init file is executing. The init file starts the bootup process and is the first process to be called when the system starts. So this runs with root privileges. Now that this is running with root privileges, this will create a file called dummy in the /system folder. In a normal situation, we cannot create a file in the system folder with normal privileges. After sending the package, we unzip the package and run the update-binary file which does the above tasks and attack is successful. We can verify it by restarting the recovery OS and logging into Android VM to find the file in /system folder.

**Task2: Inject code via app\_process**

Step 1: compile the code

Explanation: In this step, we need to create three files:

1. my\_app\_process.c

一張含有 螢幕擷取畫面, 電子用品 的圖片

自動產生的描述

1. Application.mk

一張含有 螢幕擷取畫面, 電子用品 的圖片

自動產生的描述

1. Android.mk

一張含有 螢幕擷取畫面, 電子用品, 監視器 的圖片

自動產生的描述

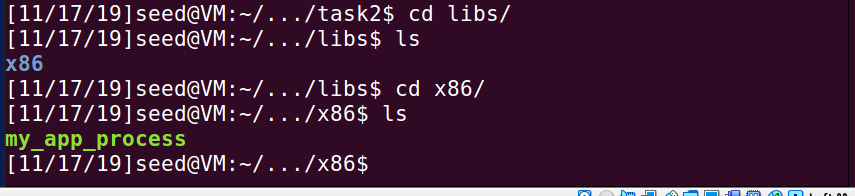
and then, we run the following commands (compile.sh) inside the source folder to compile our code. If the compilation succeeds, we can find the binary file in the ./libs/x86 folder.

一張含有 螢幕擷取畫面, 監視器, 電子用品, 顯示 的圖片

自動產生的描述

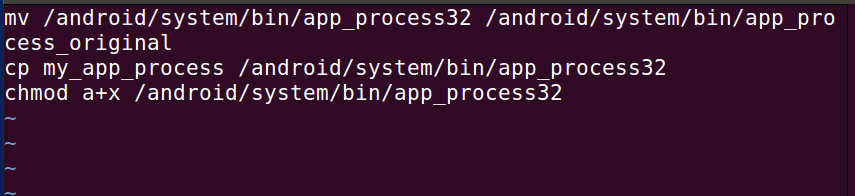
一張含有 螢幕擷取畫面 的圖片

自動產生的描述



Step 2 Write the update script and build OTA package.

Update script:



And zip the task2.

一張含有 文字, 瓶 的圖片

自動產生的描述

Send to the recovery OS of android, unzip the file and execute the update file like previous task.

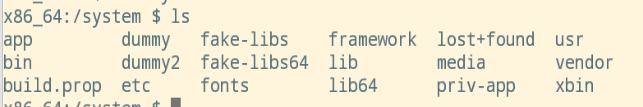
一張含有 螢幕擷取畫面, 文字 的圖片

自動產生的描述



Result: The above screenshot shows that dummy2 file is created in system folder and our

attack is successful.



Explanation: When Android starts, it always runs a program called my\_app\_process after init using root privilege. So this my\_app\_process starts the zygote daemon whose work is to start applications and this is the parent of all app processes. So we modify the my\_app\_process and it will launch something of our choice along with launching the zygote process. So we create the OTA package by creating the update-binary in the required folder hierarchy. The update-binary file will rename the app\_process32 file into something else say my\_app\_process\_original and then move the file we created into the desired location, give it executable permission, and then replace this as the new app\_process32. The file we created is compiled in such a way that it can run on any system. The app\_process32 we created will internally call the original app\_process32 now called as app\_process\_original. When we run the update-binary script, the attack is successful as seen above and the dummy2 file is created in the system folder with root permission.

**Task3: Implement SimpleSU for Getting Root Shell**

1. We used the file from seedlab called SimpleSU, unzip this file

一張含有 相片, 瓶 的圖片

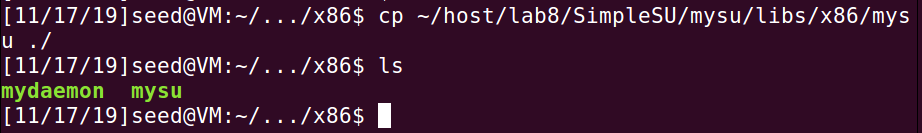
自動產生的描述

1. Run the compile\_all.sh to compile

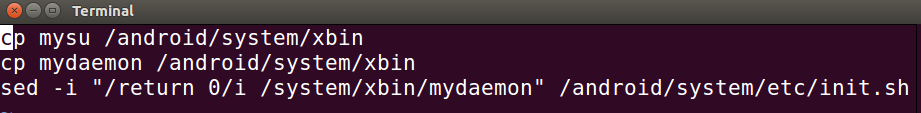
一張含有 文字, 室外 的圖片

自動產生的描述

1. Then, we got mydaemon and mysu this two files and copy to our task3 folder.



1. Modify update\_binary like following:



1. zip the files

一張含有 文字 的圖片

自動產生的描述

1. send to recovery OS of android

一張含有 螢幕擷取畫面 的圖片

自動產生的描述

1. As we can see, it shows that that mysu and mydaemon are created in the /system/xbin folder and when we execute the mysu file, we get root shell.

一張含有 螢幕擷取畫面 的圖片

自動產生的描述

Explanation: Here we want to start a root daemon so that we get a root shell. So when users want to get a root shell, they have to run a client program, which sends a request to the root daemon. Upon receiving a request, the root daemon starts a shell process and returns it to the client. The user will now have root privileges. So if users want to control the shell process, they have to be able to control the standard input and output devices of the shell process. Unfortunately, when the shell process is created, it inherits its standard input and output devices from its parent process, which is owned by root, so they are not controllable by the user’s client program. We give the client program’s output and input to the shell process, so they become the input/output devices for the shell process. In this way, the user now has complete control of the shell process.

Questions:

• Server launches the original app process binary

Filename: mydaemonsu.c Function: main() Line:252

一張含有 文字 的圖片

自動產生的描述

• Client sends its FDs

Filename: mysu.c Function: connect\_daemon() Line:101

一張含有 文字 的圖片

自動產生的描述

• Server forks to a child process

Filename: mydaemonsu.c Function: main() Line:245

一張含有 文字 的圖片

自動產生的描述

• Child process receives client’s FDs

Filename: mydaemonsu.c Function: child\_process() Line:147

一張含有 文字 的圖片

自動產生的描述

• Child process redirects its standard I/O FDs

Filename: mydaemonsu.c Function: child\_process() Line:151

一張含有 文字 的圖片

自動產生的描述

• Child process launches a root shell

Filename: mysu.c Function: main() Line:138

一張含有 文字 的圖片

自動產生的描述