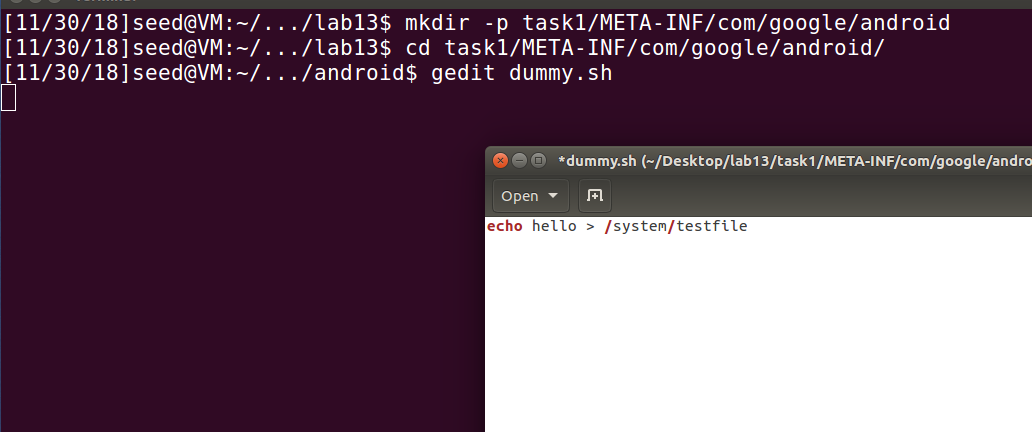
Android Device Rooting Lab

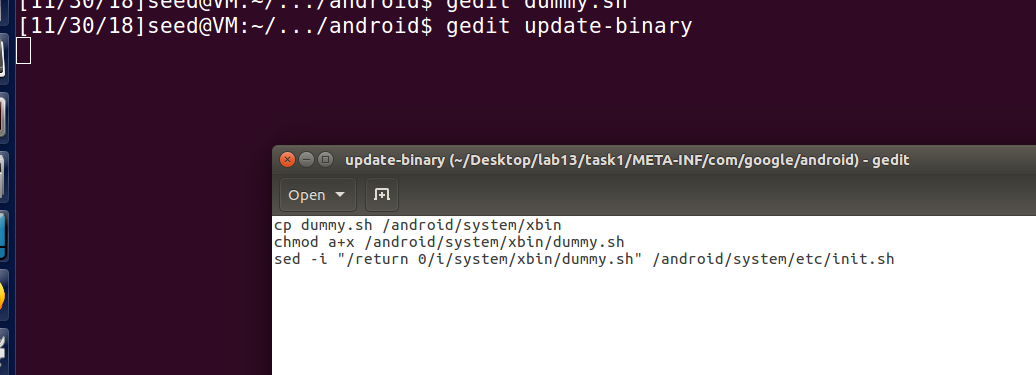
## Task 1: Build a simple OTA package

### Step 1: Write the update script

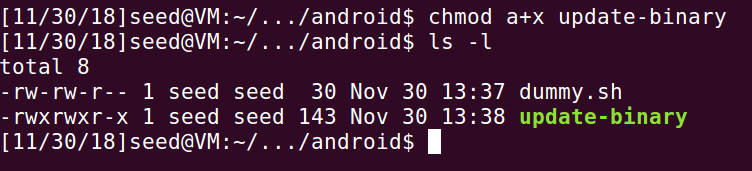
In linux OS, we create the file folder structure and write in the dummy.sh



Edit the update-binary file

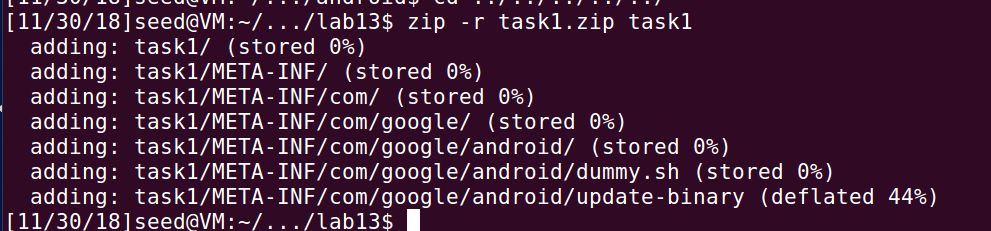


Make it runnable



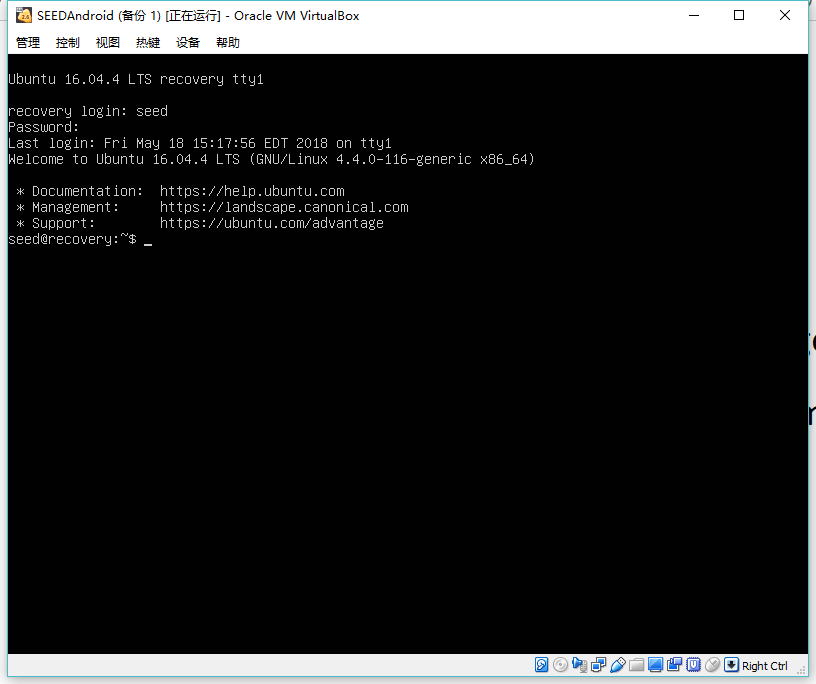
### Step 2: Build the OTA Package

Pack the folder:



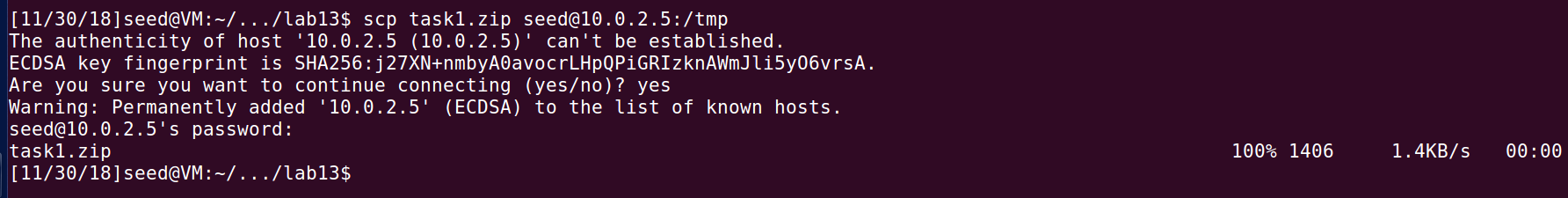
### Step 3: Run the OTA Package

Login to the Recovery Mode.

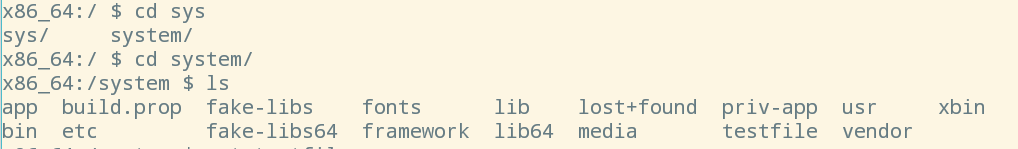


Copy OTA to Recovery OS:

scp task1.zip [seed@10.0.2.5:/tmp](mailto:seed@10.0.2.5:/tmp)





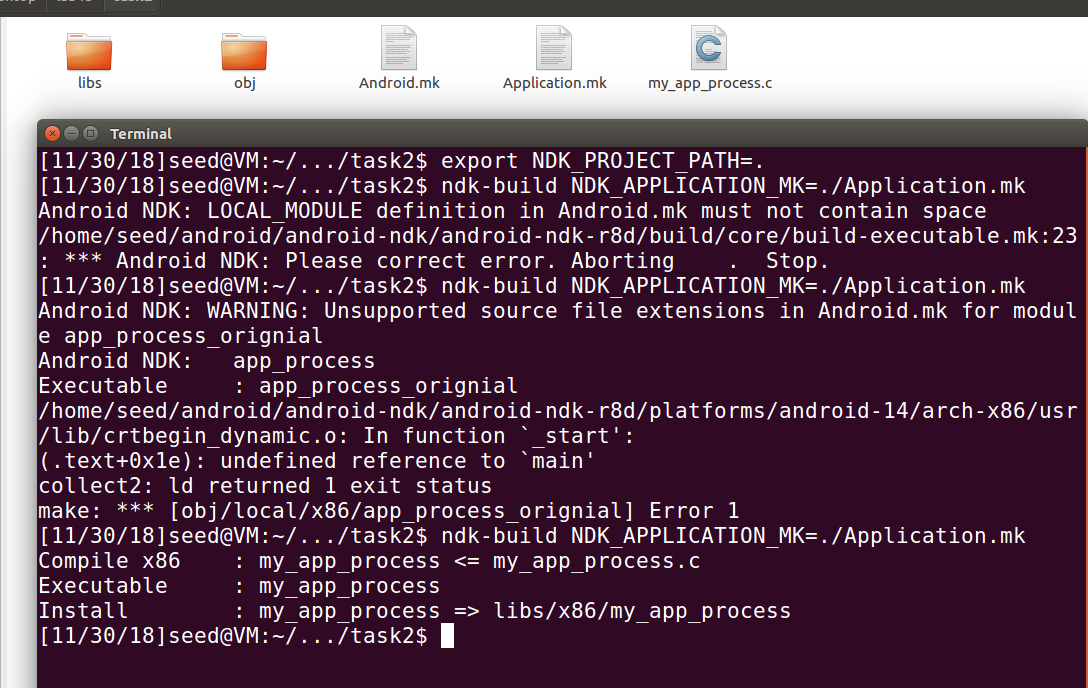


We can see that the file “testfile” is written in system folder correctly. Without the root privilege, we cannot create files under system folder.

The dummy.sh file needs to be executed automatically when Android boots up with root privilege. In this section, we know that the Android starts after the Linux system initialization. And here, we use one approach which is adding operations of init.sh. Science init.sh is executed using root privilege, the dummy program is executed correctly.

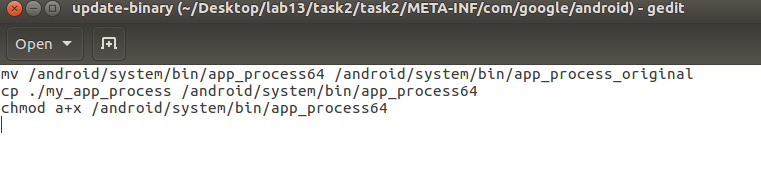
## Task 2: Inject code via app process

Compile the given code：



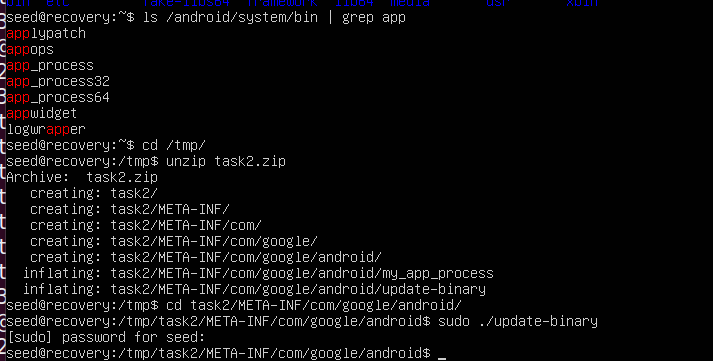
mkdir -p task2/META-INF/com/google/android

cp libs/x86/my\_app\_process task2/META-INF/com/google/android/

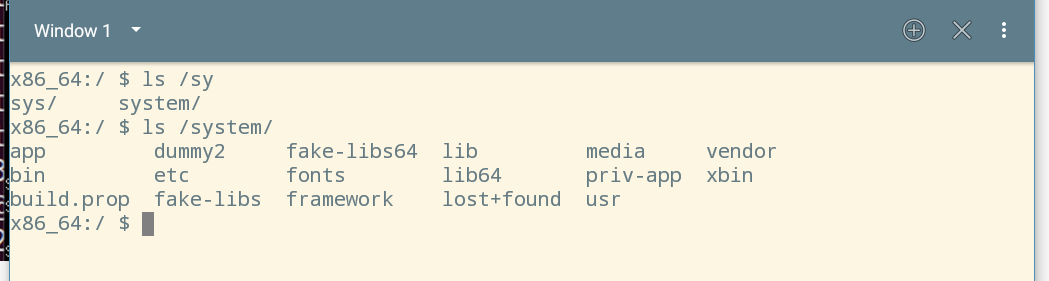


zip -r task2.zip task2

scp task2.zip seed@10.0.2.5:/tmp



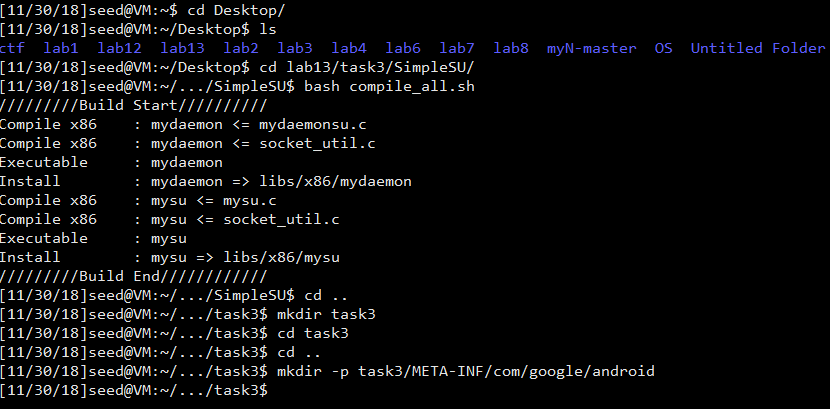
Reboot the Android VM and check the system folder:



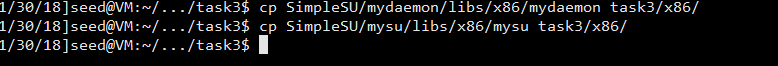
We can find that the file is created correctly.

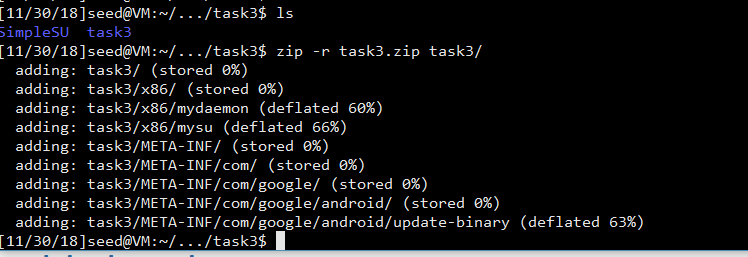
In this approach, the main idea is using a program called app\_process using the root privilege. In this approach, before the program creates the Zygote daemon, we use the program to do root-level operations. So, we compile the code under Ubuntu and run it in the Recovery Mode of Android. After executing the script file, the origin app\_process is replaced by our own program. Thus, the dummy program can be executed successfully.

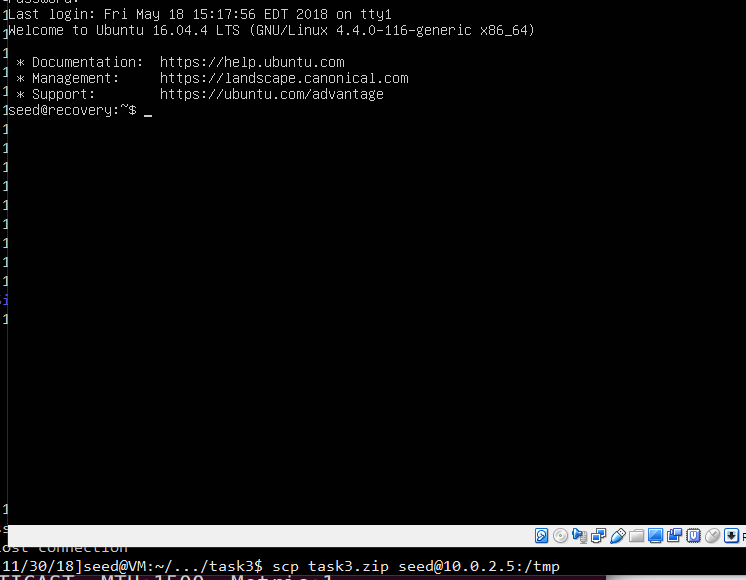
## Task 3: Implement SimpleSU for Getting Root Shell



Compile the file and copy to the folder









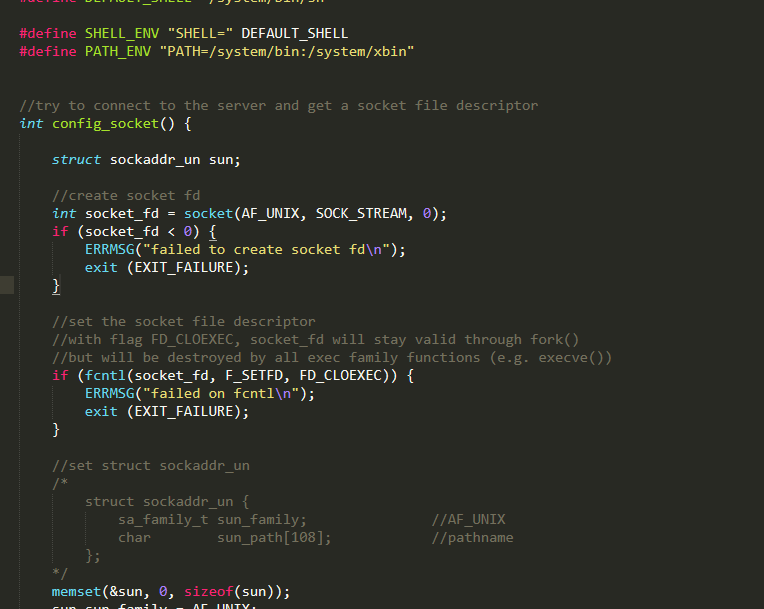
After running the update-binary code, we need to restart the VM and see what happens in the Android mode.

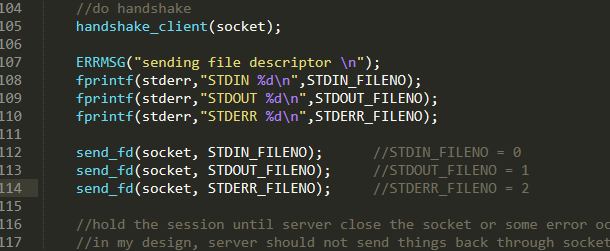


As we can see, we get the root privilege after running mysu. The result shows that the attack is correct.

(1) how to send the standard input/output devices (file descriptors) to another process

To use the socket struct. Using sockets is a common way for different process to communicate with each other. For the input and output devices, we firstly connect to a server to get the socket needed. The following two screen shot shows where does the fd come from.

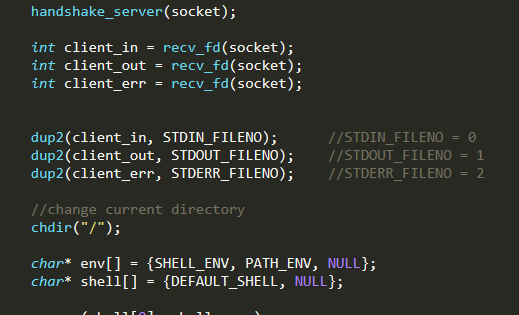




As we can see, the remote file descriptor is gotten from the server file mysu.c. This file is executed under the daemon, so it is with the root privilege. Using the socket, the rooted fd is sent.

(2) once a process receives the file descriptors, how it can use them as its input/output devices

To use them as input and output, firstly we get the original input and output. After that, we use the dup2 function to copy the rooted fd we got to link to the original fd.



Thus, the original fd is replaced with a higher “ticket”. Both input and output are gotten from the daemon using root privilege. That is why after the attack succeeded we gain a rooted shell.