

To Dr. Young Park
From : Nanthana Thanonklin
Roger Kuo
Date : May 8, 2023
Subject: CMPE 209 Team Project - Android Device Rooting Lab

Lab Setup

1. Download and install Virtualbox. Virtualbox can be downloaded from <https://www.virtualbox.org/>
 2. Download pre-built Ubuntu 16.04 from <https://seedsecuritylabs.org/labsetup.html>.
 3. Download the SEEDAndroid image file(.zip) from SEED lab course website. https://seedsecuritylabs.org/Labs_20.04/Mobile/
 4. Create new virtual machines in Virtual Box. We need to create one VM for SEEDUbuntu 16.04 and another one for SEEDAndroid.
- User manual can be found from https://seedsecuritylabs.org/Labs_20.04/Mobile/

NOTE: For this lab, we have to increase the number of the processors in Android VM to two or more, otherwise, you might see kernel panic errors and it is very important to change the graphics controller in the display settings to VBoxVGA, otherwise, you will not be able to see Android UI.

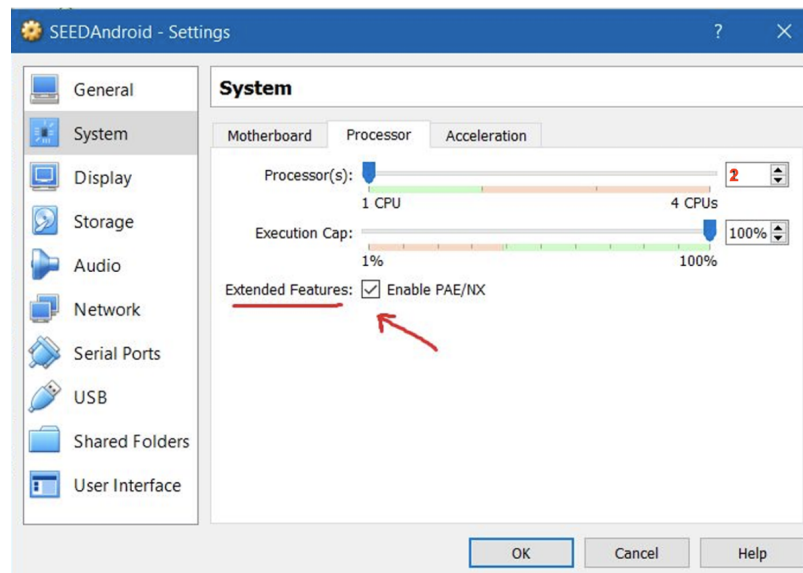
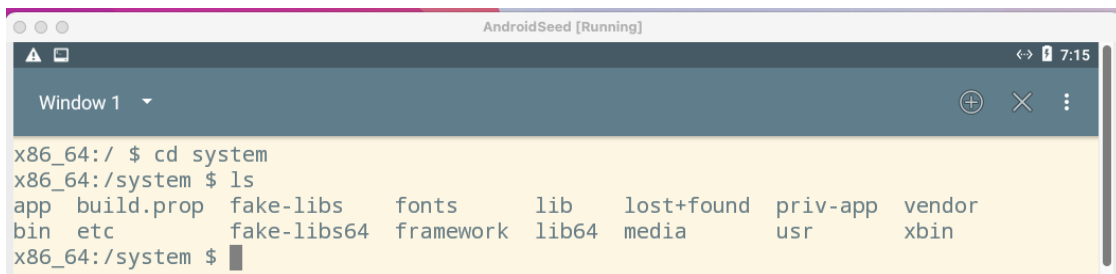


Figure 1: Android VM settings to configure CPU and PAE/NX

Lab Task 1: Build a simple OTA package

Before injecting a program into the Android OS

1. Boot up SEED Android VM, open an emulator app on the Home screen. Then, open a terminal window.
2. Navigate to the 'system' directory in the file system by typing the command "cd /system" in the terminal window.
3. Type the command ls to list all files in /system directory. Notice that it should not be a dummy.sh file in the /system directory.



```

x86_64:/ $ cd system
x86_64:/system $ ls
app  build.prop  fake-libs  fonts  lib  lost+found  priv-app  vendor
bin  etc        fake-libs64  framework  lib64  media  usr  xbin
x86_64:/system $

```

Figure 2: System directory before injecting a program into the Android OS

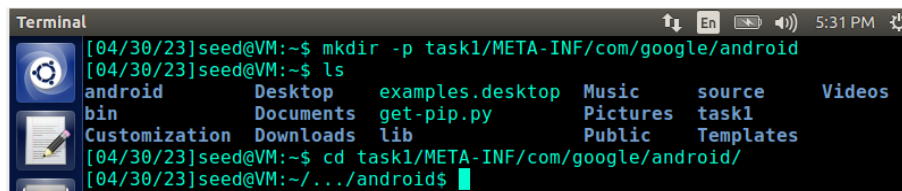
In this task, we are going to create a file called 'dummy.sh' in the '/system' folder of an Android device. However, this requires special permission (root privilege) because the '/system' folder cannot be modified by regular users. To achieve this, a command is used that places the word "hello" into the 'dummy' file. This command is stored in a script file called 'dummy.sh'.

To apply an OTA (Over-the-Air) update on an Android device, the update package file needs to be saved in the 'download' directory or the 'root' directory of the device's internal storage. In this case, we will save dummy.sh in META-INF/com/google/android.

The 'META-INF/com/google/android/update-binary' binary is responsible for executing OTA updates on an Android device. When an OTA update is initiated, this binary is loaded by the recovery operating system and it subsequently executes the 'updater-script' file that is included in the update package."

Start building OTA package

1. In Ubuntu 16.04 SEED VM, create a folder called task1, then under task1, create a subfolder according to the OTA package structure, META-INF/com/google/android/.



```

[04/30/23]seed@VM:~$ mkdir -p task1/META-INF/com/google/android
[04/30/23]seed@VM:~$ ls
android  Desktop  examples.desktop  Music  source  Videos
bin      Documents  get-pip.py        Pictures  task1
Customization  Downloads  lib              Public  Templates
[04/30/23]seed@VM:~$ cd task1/META-INF/com/google/android/
[04/30/23]seed@VM:~/.../android$

```

Figure 3: required folder structure, META-INF/com/google/android is created under task1 folder

- Open a Terminal in SEEDUbuntu 16.04 and type 'gedit dummy.sh'. Then, add 'echo hello > /system/dummy' to dummy.sh, save and close the file. Finally, type the command 'cat dummy.sh' to verify that the file contains the correct contents.

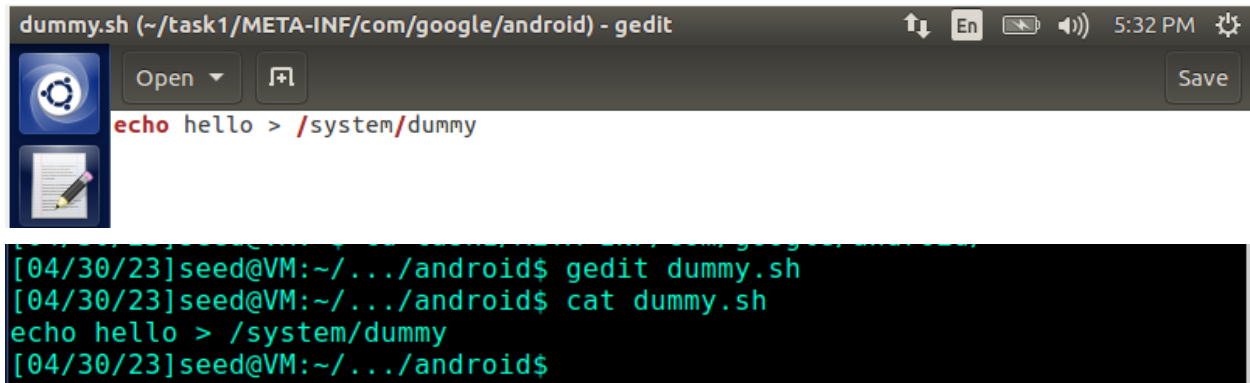


Figure 3: dummy.sh saved in task1/META-INF/com/google/android

Step 1. Write the update script

- Create the update-binary file and save it in task1/META-INF/com/google/android, same directory as dummy.sh.

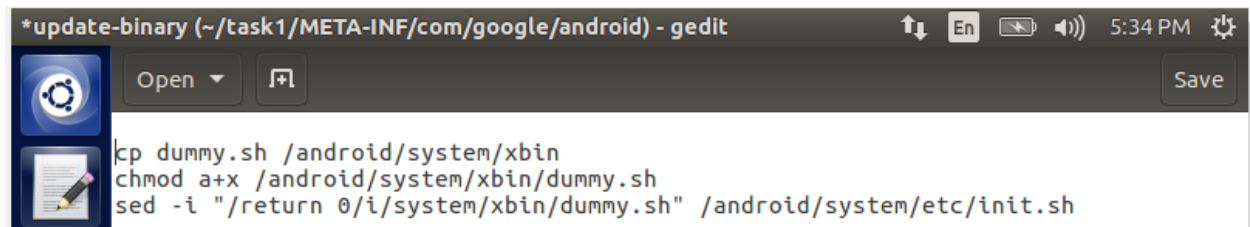


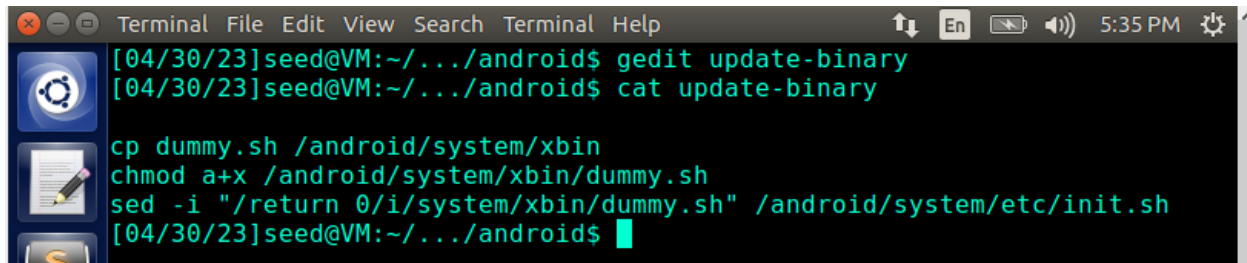
Figure 5: The contents of update-binary using gedit text editor

Line 1 - `cp dummy.sh /android/system/xbin` copies the file 'dummy.sh' to the '/android/system/xbin' directory on the Android device.

Line 2 - `chmod a+x /android/system/xbin/dummy.sh` sets the permissions on the 'dummy.sh' file to allow it to be executed. The 'chmod' command is short for 'change mode', and the 'a+x' argument specifies that all users should have execute permission on the file.

Line 3 - `sed -i "/return 0/i/system/xbin/dummy.sh"` /android/system/etc/init.sh modifies the 'init.sh' script on the Android device by inserting a line that references the 'dummy.sh' file. The 'sed' command is used for text manipulation, and the '-i' argument specifies that the changes should be made to the file in-place. The '/return 0/i' argument specifies that the line should be inserted before the line that contains

'return 0', and the '/system/xbin/dummy.sh' argument specifies the path to the 'dummy.sh' file. This modification ensures that the 'dummy.sh' file is executed every time the device boots up.



```

[04/30/23]seed@VM:~/.../android$ gedit update-binary
[04/30/23]seed@VM:~/.../android$ cat update-binary

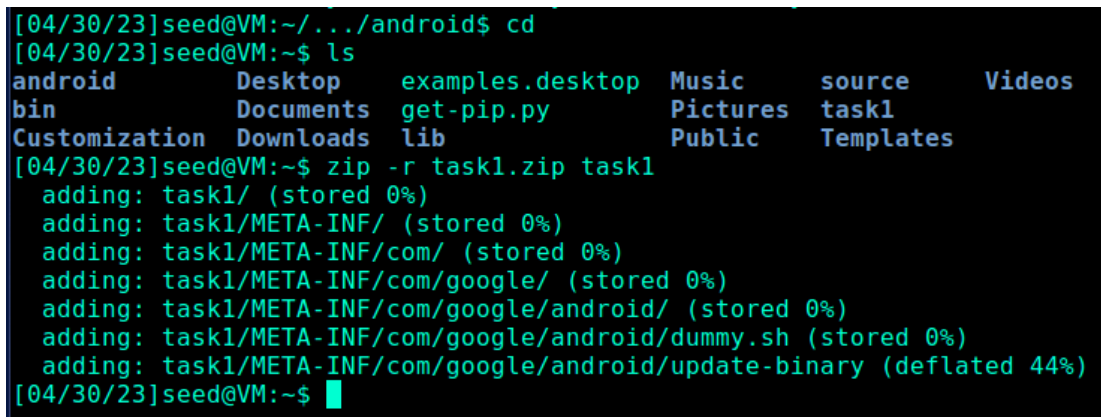
cp dummy.sh /android/system/xbin
chmod a+x /android/system/xbin/dummy.sh
sed -i "/return 0/i/system/xbin/dummy.sh" /android/system/etc/init.sh
[04/30/23]seed@VM:~/.../android$

```

Figure 6: update-binary.sh saved in task1/META-INF/com/google/android

Step 2. Build the OTA package

4. Currently, we are in task1/META-INF/com/google/android. Since we are going to zip the folder and send it to an Android VM, we are going to change to the directory where task1 is saved in. In this case, task1 is saved in /home/seed. Then, we are going to use the zip command to zip task1 by typing `zip -r task1.zip task1`



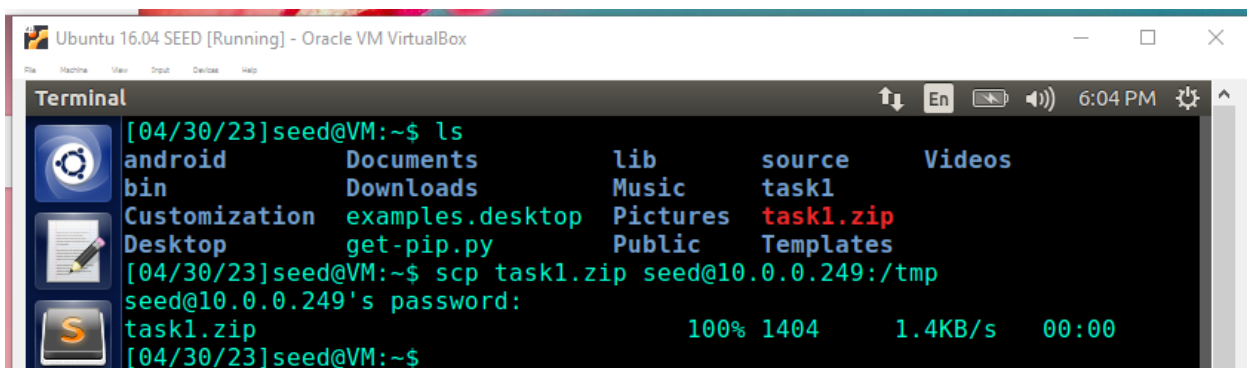
```

[04/30/23]seed@VM:~/.../android$ cd
[04/30/23]seed@VM:~$ ls
android      Desktop      examples.desktop  Music        source       Videos
bin          Documents   get-pip.py        Pictures     task1
Customization Downloads   lib               Public       Templates
[04/30/23]seed@VM:~$ zip -r task1.zip task1
adding: task1/ (stored 0%)
adding: task1/META-INF/ (stored 0%)
adding: task1/META-INF/com/ (stored 0%)
adding: task1/META-INF/com/google/ (stored 0%)
adding: task1/META-INF/com/google/android/ (stored 0%)
adding: task1/META-INF/com/google/android/dummy.sh (stored 0%)
adding: task1/META-INF/com/google/android/update-binary (deflated 44%)
[04/30/23]seed@VM:~$

```

Figure 7: zipping task1 folder

5. After finishing zipping, let's verify whether task1 has been successfully zipped by typing the command `ls`.



```

[04/30/23]seed@VM:~$ ls
android      Documents      lib          source       Videos
bin          Downloads     Music        task1
Customization examples.desktop Pictures     task1.zip
Desktop      get-pip.py    Public       Templates
[04/30/23]seed@VM:~$ scp task1.zip seed@10.0.0.249:/tmp
seed@10.0.0.249's password:
task1.zip                                100% 1404    1.4KB/s   00:00
[04/30/23]seed@VM:~$

```

Figure 8: task1.zip

6. Still on Ubuntu 16.04, we will send the zip file to the android OS using scp command. To use the scp command to send a file, we need to locate the destination IP address of the android VM.
7. To locate the IP address of the AndroidSEED VM, login to Android recovery OS, then type `ifconfig` command to obtain the required IP address. Power off the Android VM if it is still running a normal mode, then restart the VM, and quickly hold down the shift key right before you see the Virtual box logo booting up the VM to enter the GNU mode and select Ubuntu so that we can enter into Android recovery mode.

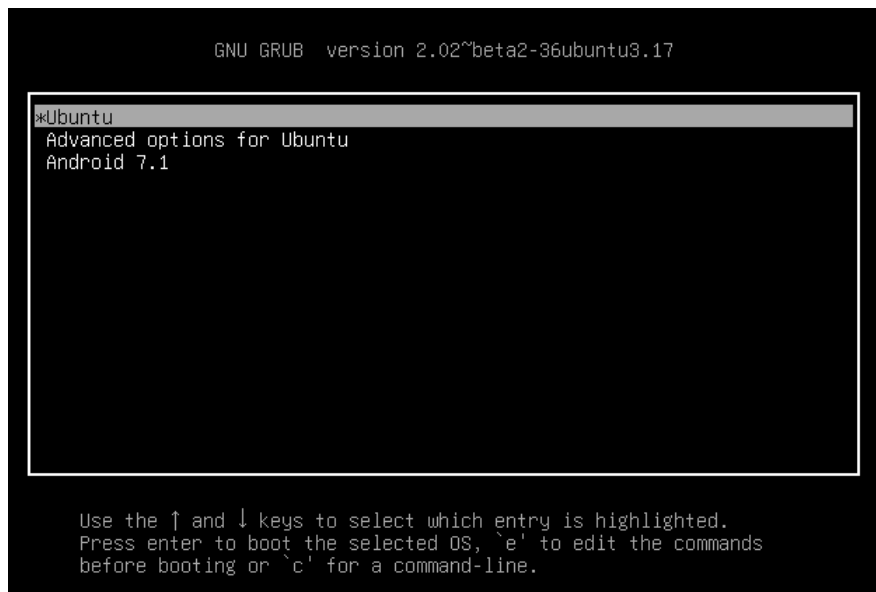


Figure 9: GNU mode used for Android Recovery Mode

8. When prompted for a recovery login, enter “seed” for the username and “dees” for the password.

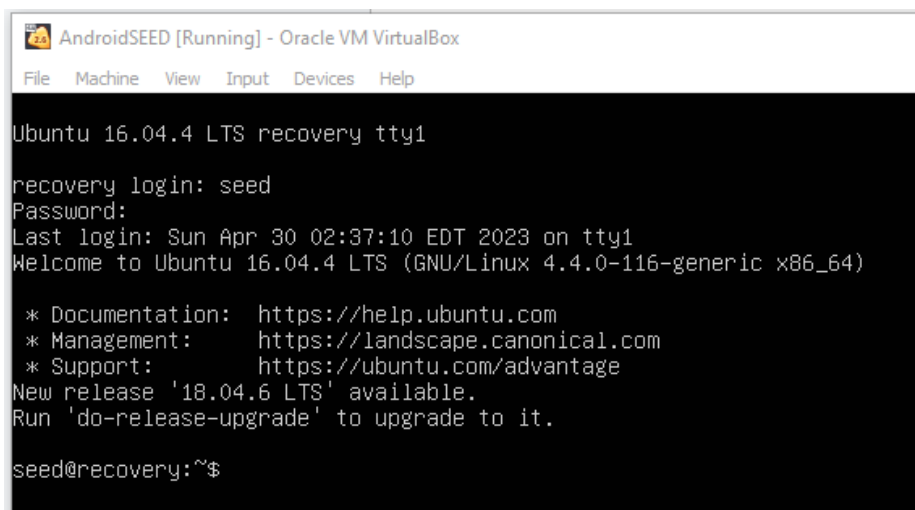
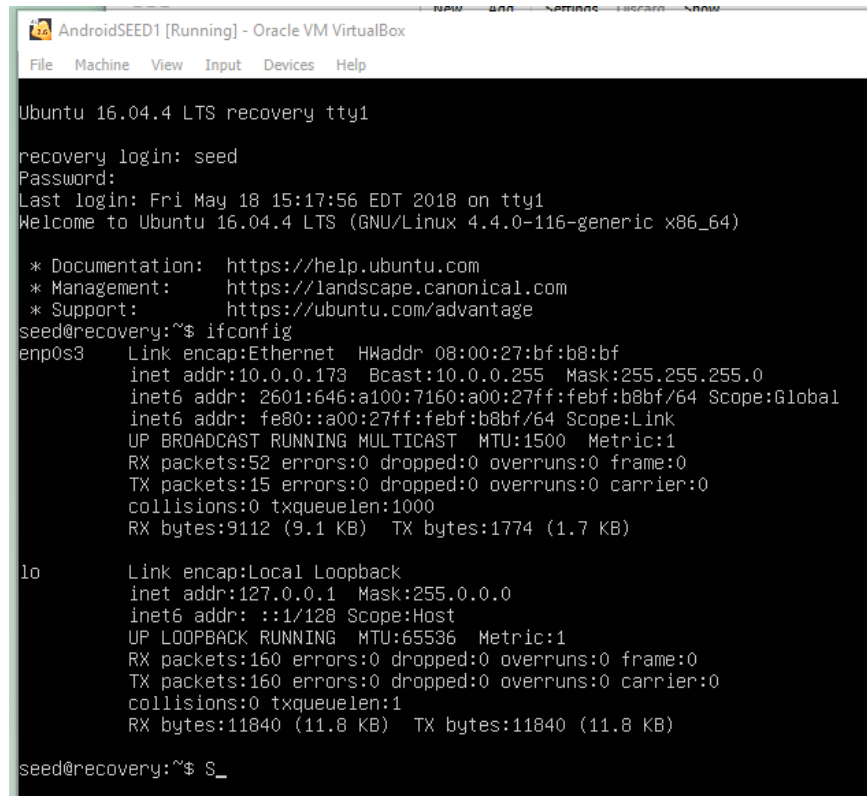


Figure 10: Android Recovery Mode



```

Ubuntu 16.04.4 LTS recovery tty1
recovery login: seed
Password:
Last login: Fri May 18 15:17:56 EDT 2018 on tty1
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-116-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage
seed@recovery:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:bf:b8:bf
          inet addr:10.0.0.173  Bcast:10.0.0.255  Mask:255.255.255.0
          inet6 addr: 2601:646:a100:7160:a00:27ff:febf:b8bf/64 Scope:Global
          inet6 addr: fe80::a00:27ff:febf:b8bf/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:52 errors:0 dropped:0 overruns:0 frame:0
          TX packets:15 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:9112 (9.1 KB)  TX bytes:1774 (1.7 KB)

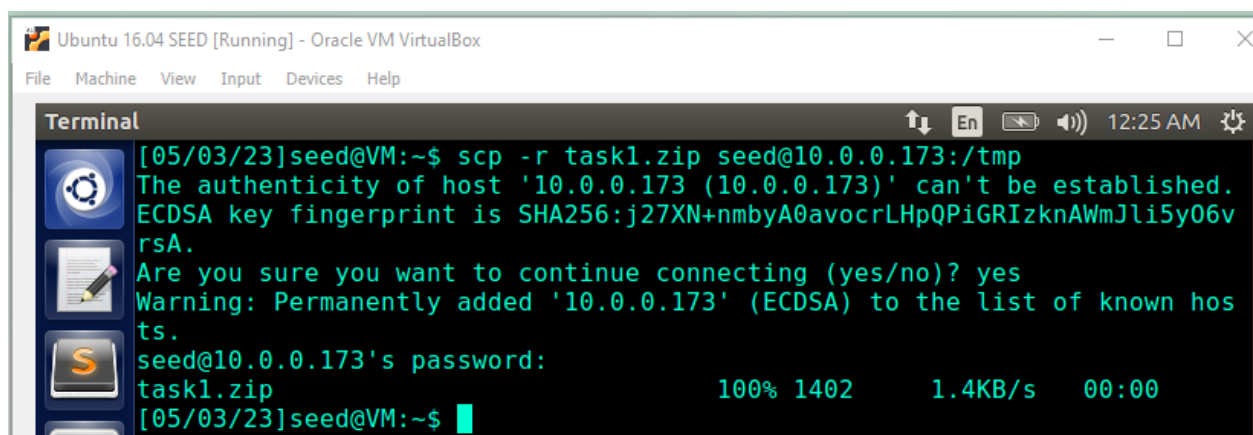
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:160 errors:0 dropped:0 overruns:0 frame:0
          TX packets:160 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:11840 (11.8 KB)  TX bytes:11840 (11.8 KB)

seed@recovery:~$ S_

```

Figure 11: IP address obtained from the Android Recovery OS

9. In Ubuntu 16.04 SEED VM, type the command, `scp -r task1.zip seed@10.0.0.249:/tmp` to transfer task1.zip to the AndroidSEED VM placing it in /tmp folder.



```

[05/03/23]seed@VM:~$ scp -r task1.zip seed@10.0.0.173:/tmp
The authenticity of host '10.0.0.173 (10.0.0.173)' can't be established.
ECDSA key fingerprint is SHA256:j27XN+nmbyA0avocrLHpQPiGRizknAWmJli5y06v
rsA.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.0.0.173' (ECDSA) to the list of known hos
ts.
seed@10.0.0.173's password:
task1.zip                               100% 1402    1.4KB/s   00:00
[05/03/23]seed@VM:~$

```

10. Since we have the Android Recovery OS window opened, we can navigate to /tmp folder to verify that the Android Recovery OS has received task1.zip sent from another OS.

```

seed@recovery:~$ ls
seed@recovery:~$ cd /tmp
seed@recovery:/tmp$ ls -la
total 36
drwxrwxrwt  8 root root 4096 Apr 30 17:55 .
drwxr-xr-x 23 root root 4096 Mar 26  2018 ..
drwxrwxrwt  2 root root 4096 Apr 30 17:52 .font-unix
drwxrwxrwt  2 root root 4096 Apr 30 17:52 .ICE-unix
drwx----- 3 root root 4096 Apr 30 17:52 systemd-private-85d218f63b664d319a24
imesyncd.service-ZuIMnc
-rw-rw-r--  1 seed seed 1404 Apr 30 17:55 task1.zip
drwxrwxrwt  2 root root 4096 Apr 30 17:52 .Test-unix
drwxrwxrwt  2 root root 4096 Apr 30 17:52 .X11-unix
drwxrwxrwt  2 root root 4096 Apr 30 17:52 .XIM-unix
seed@recovery:/tmp$

```

Figure 12: task1.zip displayed in /tmp folder of Android VM

11. In AndroidSEED VM, type unzip task1.zip

```

-rw-rw-r--  1 seed seed 1404 Apr 30 17:55 task1.zip
drwxrwxrwt  2 root root 4096 Apr 30 17:52 .Test-unix
drwxrwxrwt  2 root root 4096 Apr 30 17:52 .X11-unix
drwxrwxrwt  2 root root 4096 Apr 30 17:52 .XIM-unix
seed@recovery:/tmp$ unzip task1.zip
Archive:  task1.zip
  creating: task1/
  creating: task1/META-INF/
  creating: task1/META-INF/com/
  creating: task1/META-INF/com/google/
  creating: task1/META-INF/com/google/android/
  extracting: task1/META-INF/com/google/android/dummy.sh
  inflating: task1/META-INF/com/google/android/update-binary
seed@recovery:/tmp$

```

Figure 13: unzipping task1.zip in /tmp folder of Android VM

Step 3. Run the OTA package

12. Task1.zip has been successfully unzipped. Change the directory to where we have dummy.sh and update-binary by typing the command,
`cd /task1/META-INF/com/google/android`, then the command `ls` to list all files in that folder.

```

seed@recovery:/tmp/task1/META-INF/com/google/android$ ls
dummy.sh  update-binary
seed@recovery:/tmp/task1/META-INF/com/google/android$ chmod a+x update-binary
seed@recovery:/tmp/task1/META-INF/com/google/android$ sudo ./update-binary
seed@recovery:/tmp/task1/META-INF/com/google/android$ sudo reboot

```

Figure 14: The contents of task1, dummy.sh and update-binary

13. Execute update-binary script so that dummy.sh will be written in the /system folder.
Then, reboot.

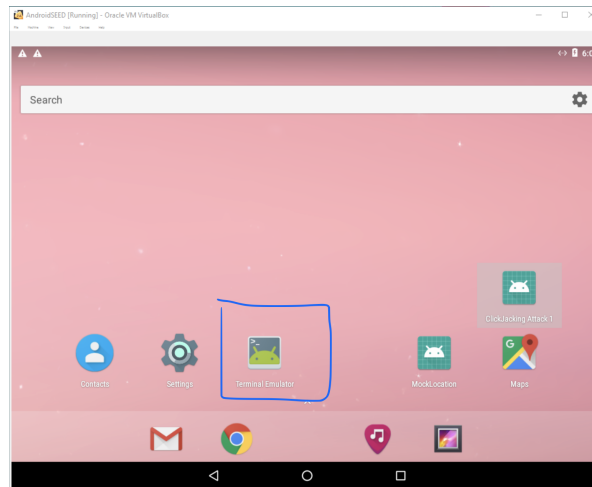


Figure 15: Android GUI mode - Terminal Emulator

14. To verify that we have successfully gained access to the root, login to Android VM, then open the Terminal Emulator and type `cd /system`.

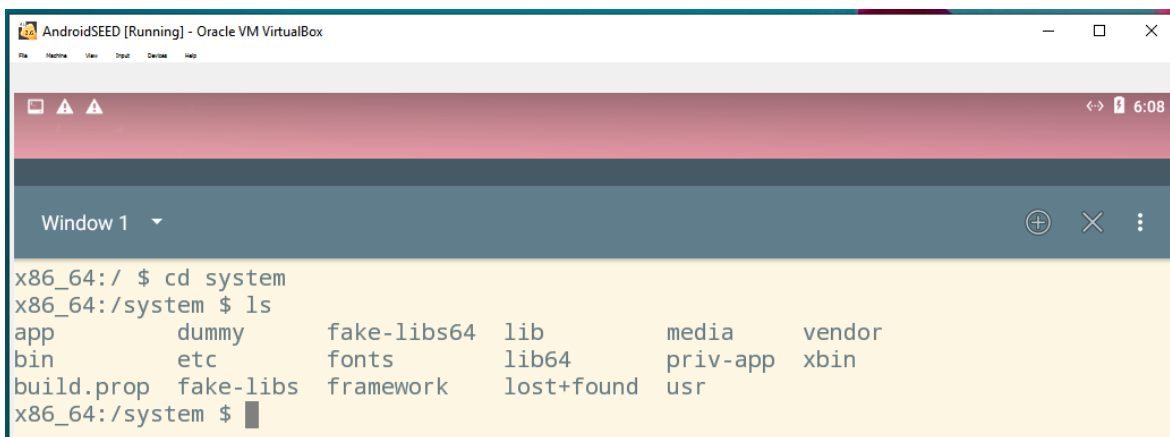


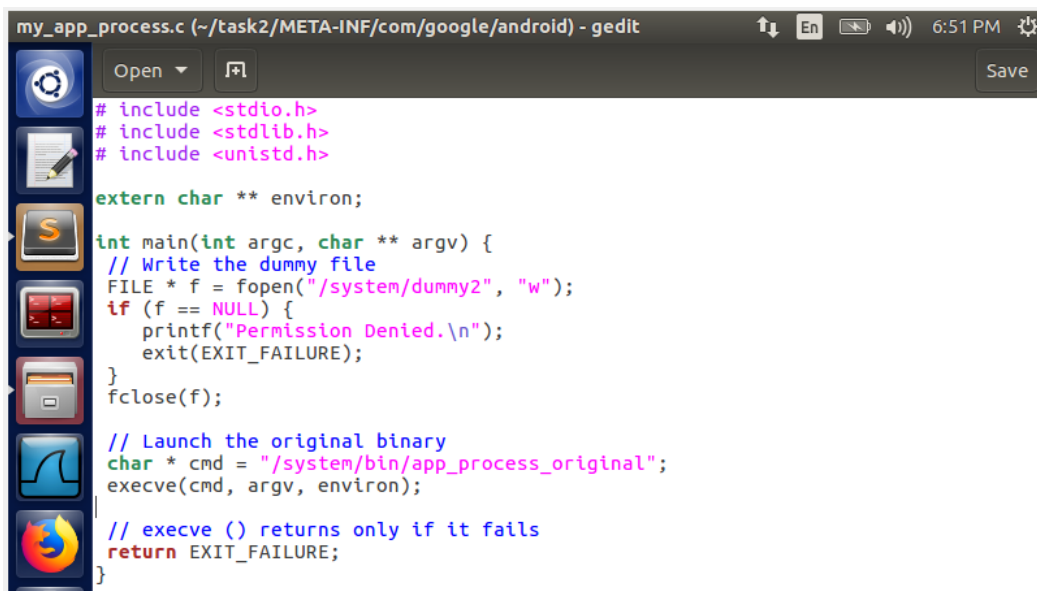
Figure 16: Android GUI mode - dummy file displayed in /system folder

In this lab, we successfully built a simple OTA package by creating and modifying files in the Android system. We started by creating a dummy file in the /system folder using root privilege, then added the content of dummy.sh using gedit in Terminal. We then created an update-binary file and saved it in task1/META-INF/com/google/android. After zipping the folder and sending it to an Android VM, we updated the recovery OS using the OTA package to gain root access. Overall, the lab was a success in demonstrating the process of building a simple OTA package.

Lab Task 2: Inject code via app_process

When the Android runtime initializes, it executes an application process named "app process" with root privileges. The purpose of this process is to start the Zygote daemon, which is responsible for launching applications. As a result, Zygote serves as the parent process for all application processes. Our objective is to alter the behavior of app process such that, in addition to starting the Zygote daemon, it also runs a custom program of our choosing. As in the previous task, we aim to demonstrate that we can execute our program with root privileges by placing a dummy file (dummy2) in the /system folder.

The code provided below serves as a wrapper for the original app process. To implement this wrapper, we will rename the original app process binary to "app process original" and then use "app process" as the name for our wrapper program. Within our wrapper code, we first write some data to the dummy file and then execute the original app process.



```
my_app_process.c (~/task2/META-INF/com/google/android) - gedit
# include <stdio.h>
# include <stdlib.h>
# include <unistd.h>

extern char ** environ;

int main(int argc, char ** argv) {
    // Write the dummy file
    FILE * f = fopen("/system/dummy2", "w");
    if (f == NULL) {
        printf("Permission Denied.\n");
        exit(EXIT_FAILURE);
    }
    fclose(f);

    // Launch the original binary
    char * cmd = "/system/bin/app_process_original";
    execve(cmd, argv, environ);

    // execve () returns only if it fails
    return EXIT_FAILURE;
}
```

Figure 17: my_app_process.c

Code Explanation

LOCAL_PATH := \$(call my-dir) sets a variable LOCAL_PATH to the current directory. The include \$(CLEAR_VARS) command clears all variables defined previously by the Android.mk file.

The LOCAL_MODULE variable specifies the name of the module to be built, which in this case is my_app_process.

The `LOCAL_SRC_FILES` variable specifies the source files needed to build the module, which in this case is `my_app_process.c`.

The `include $(BUILD_EXECUTABLE)` command includes the rules for building an executable in the make system. This rule takes the values specified in `LOCAL_MODULE` and `LOCAL_SRC_FILES` and creates an executable file with the specified name

To create our wrapper program "app_process," we must use the Native Development Kit (NDK) to compile a standalone native program. It is crucial to note that we must perform this compilation process in our SEEDUbuntu virtual machine and not within the Recovery OS or Android OS, as neither of them has the necessary development environment to compile native code. Fortunately, we have installed the NDK in our SEEDUbuntu VM, which is a collection of tools that facilitate the compilation of C and C++ code for the Android OS.

Step 1. Compile the code.

1. Go to Android VM, boot into normal/GUI mode. Then, open the Terminal Emulator app.

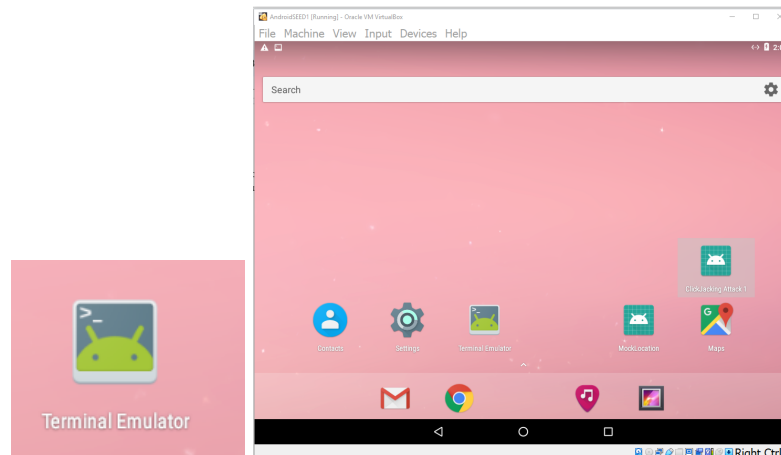


Figure 17: AndroidVM - Terminal Emulator

2. Before we begin rooting, let's verify that there is no `dummy2.sh` in the Android root or in the `/system` folder. To do this, type the command `cd /system` then `ls`. Notice that `dummy2` is not there yet.



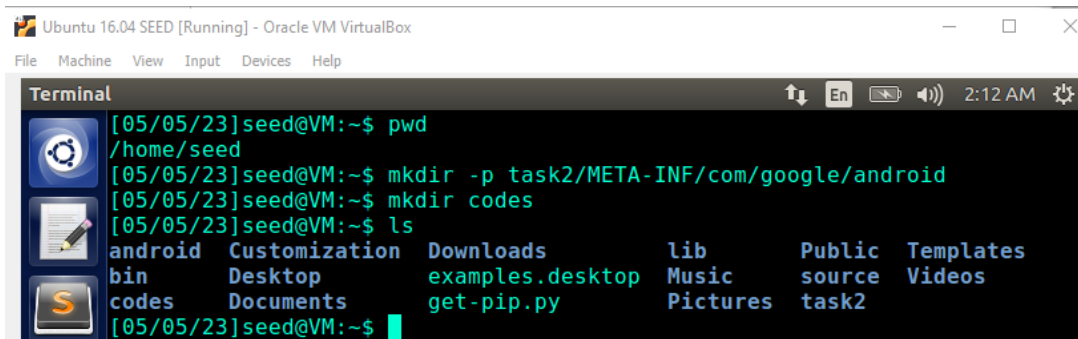
```

x86_64:/ # pwd
/
x86_64:/ # cd /system
x86_64:/system # ls
app  build.prop  fake-libs  fonts  lib  lost+found  priv-app  vendor
bin  etc        fake-libs64  framework  lib64  media  usr  xbin
x86_64:/system #

```

Figure 18: No dummy2.sh found in /system folder

3. Execute the following command in the terminal to create a folder with the required structure for Task 2, as provided in the SEED lab manual. This will enable us to zip the contents of the folder and transfer it to the Android Recovery OS at a later stage.
`mkdir -p task2/META-INF/com/google/android`
4. To utilize the NDK, we must create two files, Application.mk and Android.mk, and store them in the same directory as our source code. To achieve this, we will create a new directory called "codes," and we will generate the Application.mk and Android.mk files within this directory.



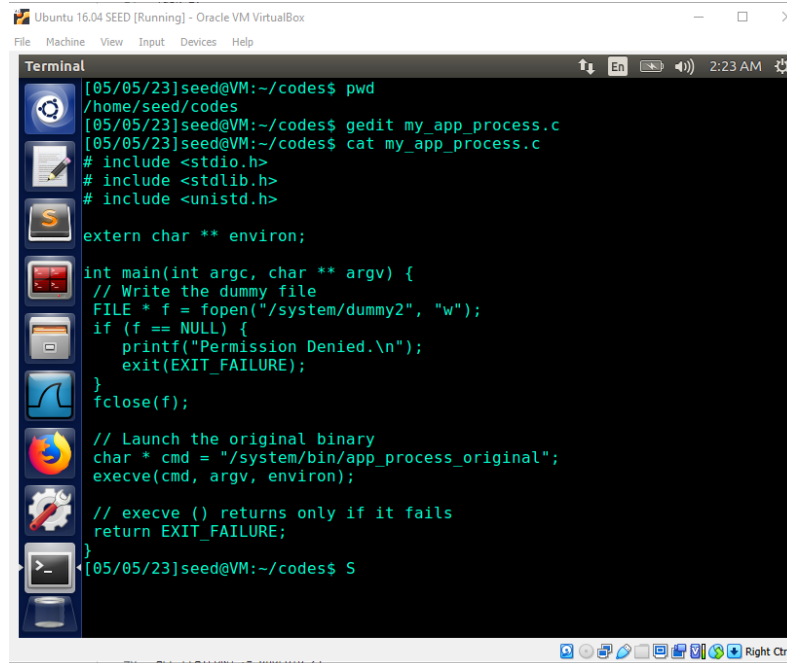
```

[05/05/23]seed@VM:~$ pwd
/home/seed
[05/05/23]seed@VM:~$ mkdir -p task2/META-INF/com/google/android
[05/05/23]seed@VM:~$ mkdir codes
[05/05/23]seed@VM:~$ ls
android  Customization  Downloads  lib  Public  Templates
bin      Desktop        examples.desktop  Music  source  Videos
codes   Documents      get-pip.py  Pictures  task2
[05/05/23]seed@VM:~$

```

Figure 19: Creating required folders for task2

5. Type the command "gedit" in the terminal to create the file "my_app_process.c". The code for this file can be found in the Seed Lab manual.



```

[05/05/23]seed@VM:~/codes$ pwd
/home/seed/codes
[05/05/23]seed@VM:~/codes$ gedit my_app_process.c
[05/05/23]seed@VM:~/codes$ cat my_app_process.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

extern char ** environ;

int main(int argc, char ** argv) {
    // Write the dummy file
    FILE * f = fopen("/system/dummy2", "w");
    if (f == NULL) {
        printf("Permission Denied.\n");
        exit(EXIT_FAILURE);
    }
    fclose(f);

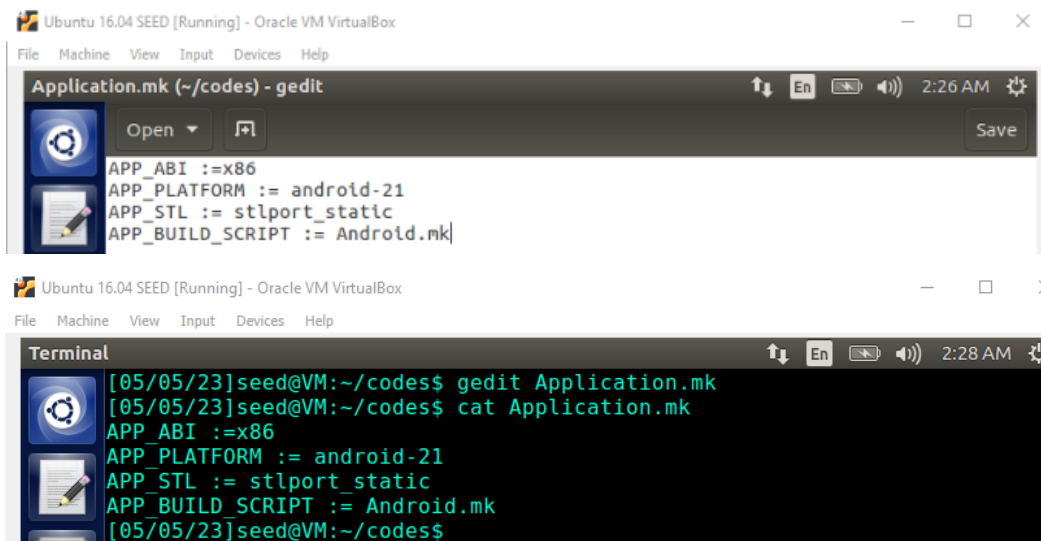
    // Launch the original binary
    char * cmd = "/system/bin/app_process_original";
    execve(cmd, argv, environ);

    // execve () returns only if it fails
    return EXIT_FAILURE;
}
[05/05/23]seed@VM:~/codes$ S

```

Figure 19: Creating my_app_process.c and save it in the codes folder

6. Type the command "gedit" in the terminal to create the file "Application.mk". The code for this file can be found in the Seed Lab manual.



```

Application.mk (~/.codes) - gedit
APP_ABI := x86
APP_PLATFORM := android-21
APP_STL := stlport_static
APP_BUILD_SCRIPT := Android.mk

Terminal
[05/05/23]seed@VM:~/codes$ gedit Application.mk
[05/05/23]seed@VM:~/codes$ cat Application.mk
APP_ABI := x86
APP_PLATFORM := android-21
APP_STL := stlport_static
APP_BUILD_SCRIPT := Android.mk
[05/05/23]seed@VM:~/codes$

```

Figure 20: Application.mk

7. Type the command "gedit" in the terminal to create the file "Android.mk". The code for this file can be found in the Seed Lab manual.

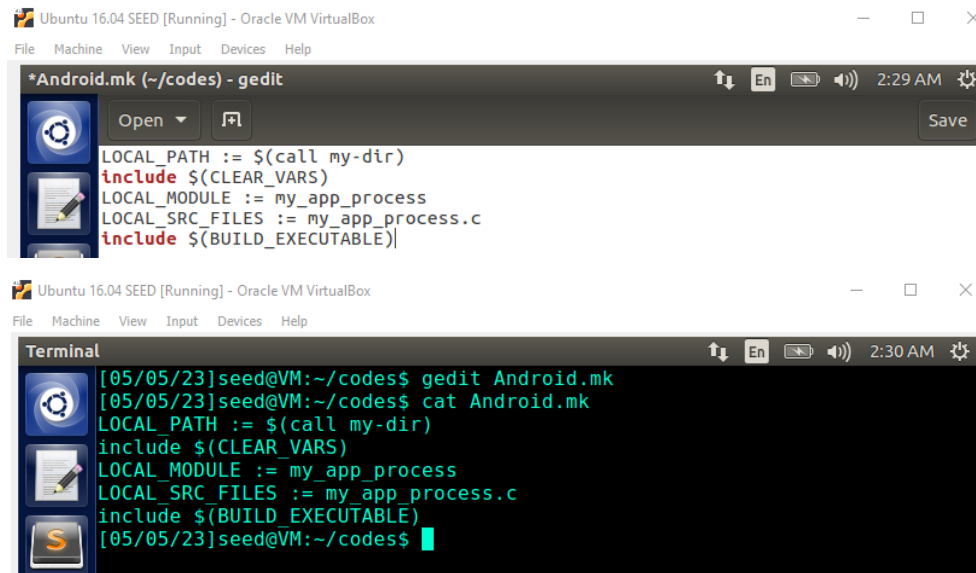


Figure 21: Android.mk

8. Execute the following commands within the source directory to compile our code. If the compilation process is successful, we will locate the binary file in the ./libs/x86 folder.

```
export NDK_PROJECT_PATH=.
ndk-build NDK_APPLICATION_MK=./Application.mk
```

9. Type the command "gedit" in the terminal to create the file "compile.sh". Add the above two lines into compile.sh.



Figure 22: compile.sh

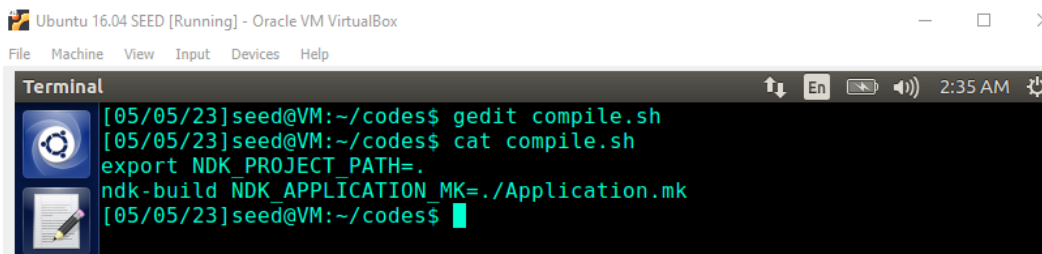
Explanation

The code `export NDK_PROJECT_PATH=.` sets an environment variable named `NDK_PROJECT_PATH` to the current directory. (i.e., the directory where this command is executed).

The NDK (Native Development Kit) is a set of tools that allow developers to write and compile native code (C/C++) for Android applications. The `ndk-build` command is used to build native code projects using the NDK.

The code `ndk-build NDK_APPLICATION_MK=./Application.mk` invokes the `ndk-build` command with an option `-NDK_APPLICATION_MK` specifying the path to the `Application.mk` file in the current directory (`./`). The `Application.mk` file is a makefile used by the NDK build system to specify build options for the project. This command builds the native code project using the options specified in the `Application.mk` file.

10. Use the command, `cat compile.sh` to display the contents of `compile.sh` to verify that the we have added the code correctly.

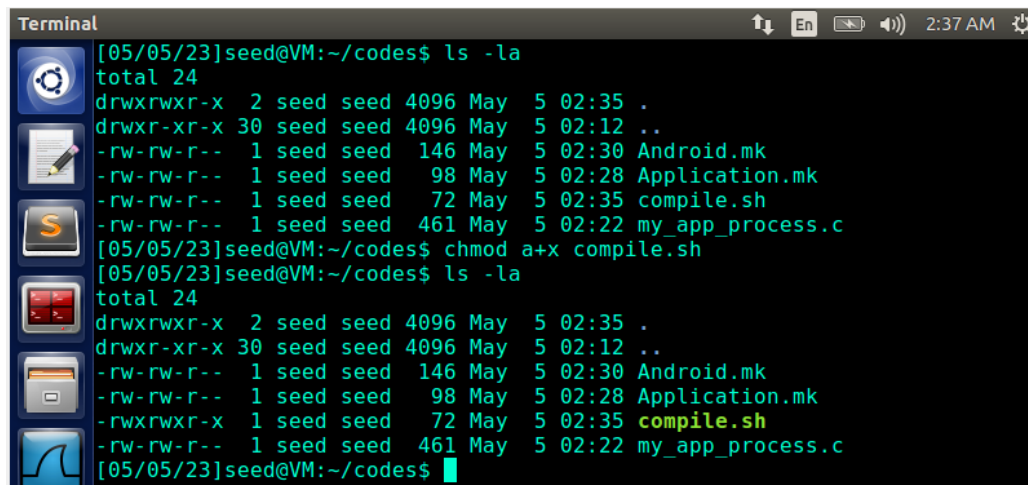


```

[05/05/23]seed@VM:~/codes$ gedit compile.sh
[05/05/23]seed@VM:~/codes$ cat compile.sh
export NDK_PROJECT_PATH=.
ndk-build NDK_APPLICATION_MK=./Application.mk
[05/05/23]seed@VM:~/codes$
  
```

Figure 23: compile.sh

11. To enable us to run the file "`compile.sh`", we must add execute permissions. Type the command "`chmod a+x compile.sh`" to achieve this.

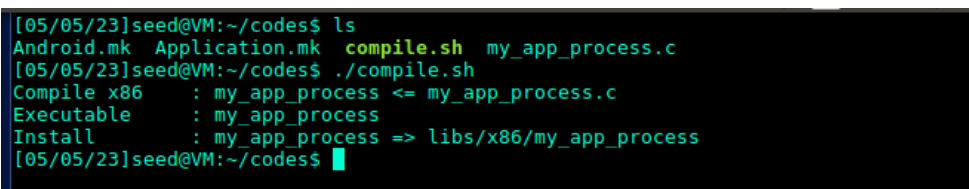


```

[05/05/23]seed@VM:~/codes$ ls -la
total 24
drwxrwxr-x  2 seed seed 4096 May  5 02:35 .
drwxr-xr-x 30 seed seed 4096 May  5 02:12 ..
-rw-rw-r--  1 seed seed  146 May  5 02:30 Android.mk
-rw-rw-r--  1 seed seed   98 May  5 02:28 Application.mk
-rw-rw-r--  1 seed seed   72 May  5 02:35 compile.sh
-rw-rw-r--  1 seed seed  461 May  5 02:22 my_app_process.c
[05/05/23]seed@VM:~/codes$ chmod a+x compile.sh
[05/05/23]seed@VM:~/codes$ ls -la
total 24
drwxrwxr-x  2 seed seed 4096 May  5 02:35 .
drwxr-xr-x 30 seed seed 4096 May  5 02:12 ..
-rw-rw-r--  1 seed seed  146 May  5 02:30 Android.mk
-rw-rw-r--  1 seed seed   98 May  5 02:28 Application.mk
-rwxrwxr-x  1 seed seed   72 May  5 02:35 compile.sh
-rw-rw-r--  1 seed seed  461 May  5 02:22 my_app_process.c
[05/05/23]seed@VM:~/codes$
  
```

Figure 24: executable compile.sh

12. Compile `compile.sh` by typing `./compile.sh`.



```


[05/05/23]seed@VM:~/codes$ ls
Android.mk  Application.mk  compile.sh  my_app_process.c
[05/05/23]seed@VM:~/codes$ ./compile.sh
Compile x86      : my_app_process <= my_app_process.c
Executable      : my_app_process
Install         : my_app_process => libs/x86/my_app_process
[05/05/23]seed@VM:~/codes$
  
```

Figure 25: Compile compile.sh

```
[05/05/23]seed@VM:~/codes$ ls -la
total 32
drwxrwxr-x  4 seed seed 4096 May  5 02:50 .
drwxr-xr-x 30 seed seed 4096 May  5 02:12 ..
-rw-rw-r--  1 seed seed 146 May  5 02:30 Android.mk
-rw-rw-r--  1 seed seed  98 May  5 02:28 Application.mk
-rwxrwxr-x  1 seed seed  72 May  5 02:35 compile.sh
drwxrwxr-x  3 seed seed 4096 May  5 02:50 libs
-rw-rw-r--  1 seed seed 461 May  5 02:22 my_app_process.c
drwxrwxr-x  3 seed seed 4096 May  5 02:50 obj
[05/05/23]seed@VM:~/codes$
```

Figure 26: libs and obj generated from compile.sh

13. The compilation process was successful, and we can locate the resulting binary file in the `./libs/x86` folder. Type the command `cd ./libs/x86` to locate my `app` process.



```
Ubuntu 16.04 SEED [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

Terminal
[05/05/23]seed@VM:~/codes$ cd libs/x86/
[05/05/23]seed@VM:~/../x86$ ls -la
total 16
drwxrwxr-x 2 seed seed 4096 May  5 02:40 .
drwxrwxr-x 3 seed seed 4096 May  5 02:40 ..
-rwxr-xr-x 1 seed seed 5116 May  5 02:40 my_app_process
[05/05/23]seed@VM:~/../x86$
```

Figure 27: compiled binary code located in /libs/x86

[illegible]

Figure 28: my app process, the compiled binary code

Step 2. Write the update script and build OTA package.

1. To proceed, we must copy our compiled binary code to the appropriate location within the Android system, so will copy `my_app_process` to `Task2/META-INF/com/google/android`.

```
[05/05/23]seed@VM:~/.../x86$ mv my_app_process ~/task2/META-INF/com/google/android/
[05/05/23]seed@VM:~/.../x86$ cd ~/task2/META-INF/com/google/android/
[05/05/23]seed@VM:~/.../android$ ls -la
total 16
drwxrwxr-x 2 seed seed 4096 May  5 02:59 .
drwxrwxr-x 3 seed seed 4096 May  5 02:10 ..
-rwxr-xr-x 1 seed seed 5116 May  5 02:51 my_app_process
[05/05/23]seed@VM:~/.../android$ pwd
/home/seed/task2/META-INF/com/google/android
[05/05/23]seed@VM:~/.../android$ S
```

Figure 29: my_app_process copied to /task2/META-INF/com/google/android

2. We need to rename the original app_process binary to a new name, and then use our compiled code as the new app_process. The name of the original app process binary can either be app_process32 or app_process64 depending on the device's architecture. Since our Android VM is a 64-bit device, we will use the name "app_process64" for our new binary.
3. Type the command "gedit" in the terminal to create the file "update-binary" in the same directory as my_app_process. The code for this file can be found in the Seed Lab manual.

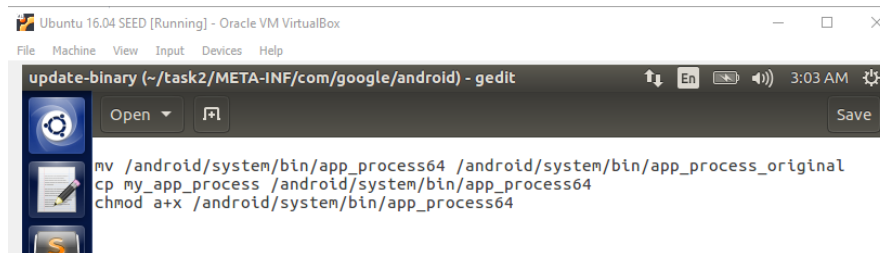


Figure 30: update_binary

4. Type the command `ls -la` to verify that we have my_app_process, the compiled binary file, not my_app_process.c file and update-binary in the same source folder, which is task2/META-INF/com/google/android.

```
[05/05/23]seed@VM:~/.../android$ cat update-binary
mv /android/system/bin/app_process64 /android/system/bin/app_process_ori
ginal
cp my_app_process /android/system/bin/app_process64
chmod a+x /android/system/bin/app_process64
[05/05/23]seed@VM:~/.../android$ ls -la
total 20
drwxrwxr-x 2 seed seed 4096 May  5 03:07 .
drwxrwxr-x 3 seed seed 4096 May  5 02:10 ..
-rwxr-xr-x 1 seed seed 5116 May  5 02:51 my_app_process
-rw-rw-r-- 1 seed seed 175 May  5 03:07 update-binary
[05/05/23]seed@VM:~/.../android$ pwd
/home/seed/task2/META-INF/com/google/android
[05/05/23]seed@VM:~/.../android$
```

Figure 31: my_app_process and update-binary in the same source folder

5. To enable us to run the file "update-binary", we must add execute permissions. Execute the command "`chmod a+x update-binary.sh`" to achieve this.


```
[05/05/23]seed@VM:~/../android$ ls -la
total 20
drwxrwxr-x 2 seed seed 4096 May  5 03:07 .
drwxrwxr-x 3 seed seed 4096 May  5 02:10 ..
-rwxr-xr-x 1 seed seed 5116 May  5 02:51 my_app_process
-rw-rw-r-- 1 seed seed  175 May  5 03:07 update-binary
[05/05/23]seed@VM:~/../android$ chmod a+x update-binary
[05/05/23]seed@VM:~/../android$ ls -la
total 20
drwxrwxr-x 2 seed seed 4096 May  5 03:07 .
drwxrwxr-x 3 seed seed 4096 May  5 02:10 ..
-rwxr-xr-x 1 seed seed 5116 May  5 02:51 my_app_process
-rwxrwxr-x 1 seed seed  175 May  5 03:07 update-binary
[05/05/23]seed@VM:~/../android$
```

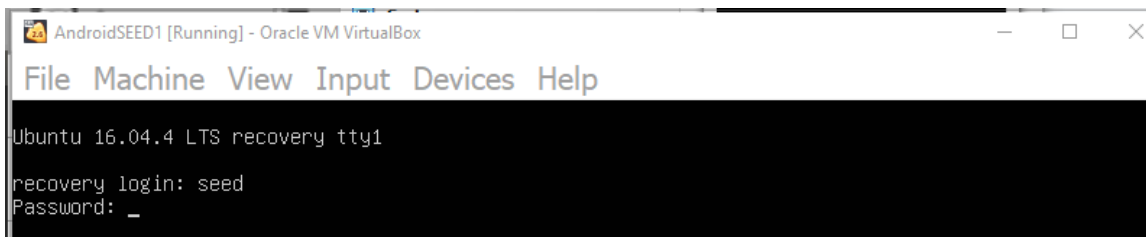
Figure 32: executable update-binary

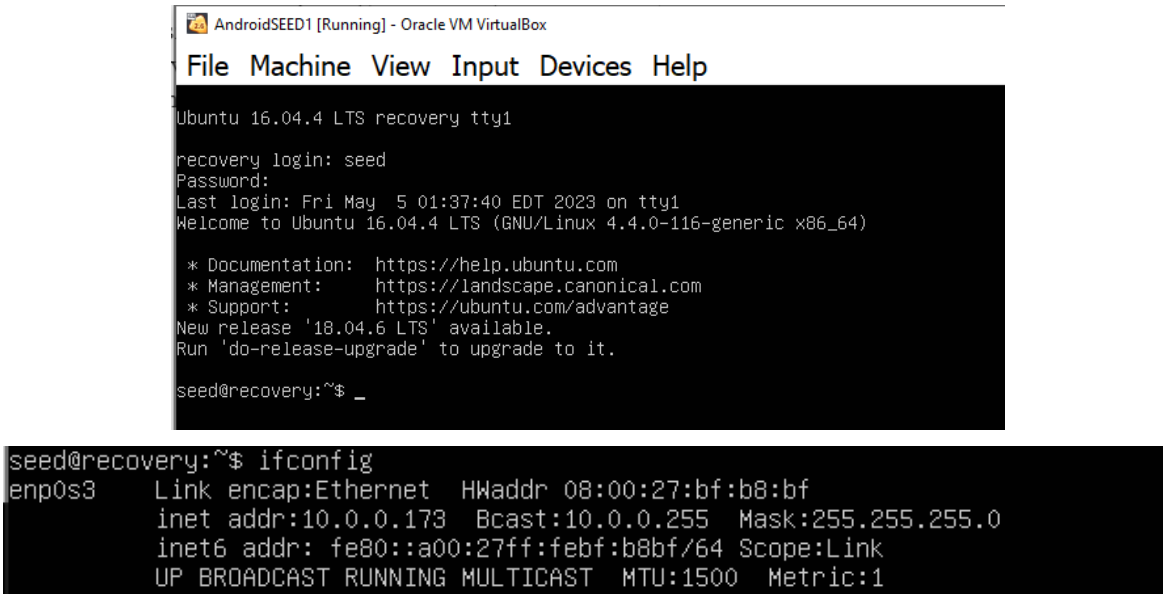
- Change the directory to `/home/seed`, then zip task2 folder by typing the command `zip -r task2.zip task2`.

```
[05/05/23]seed@VM:~/../android$ cd /home/seed/
[05/05/23]seed@VM:~$ ls
android  Customization  Downloads      lib           Public  Templates
bin      Desktop        examples.desktop  Music        source  Videos
codes   Documents      get-pip.py      Pictures      task2
[05/05/23]seed@VM:~$ zip -r task2.zip task2
adding: task2/ (stored 0%)
adding: task2/META-INF/ (stored 0%)
adding: task2/META-INF/com/ (stored 0%)
adding: task2/META-INF/com/google/ (stored 0%)
adding: task2/META-INF/com/google/android/ (stored 0%)
adding: task2/META-INF/com/google/android/update-binary (deflated 58%)
adding: task2/META-INF/com/google/android/my_app_process (deflated 72%)
[05/05/23]seed@VM:~$
```

Figure 33: zipping task2 folder

- Upon successfully building the OTA package, we can deliver it to the recovery OS. In a real-world scenario, the recovery OS would automatically execute the package. However, since we are operating within a lab environment, we are using Ubuntu as our recovery OS. Ubuntu, however, does not possess the necessary recovery functionality. Therefore, we must emulate the recovery functionality ourselves by manually unpacking the OTA package (using the `unzip` command). To deliver the zipped task2 folder, type the command `scp -r task2.zip seed@IPofrecoveryOS:\tmp`.
- Login to Android Recovery OS by holding down the shift key before the system is booting up. Use the arrow up to select “Ubuntu”





```

AndroidSEED1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

Ubuntu 16.04.4 LTS recovery tty1

recovery login: seed
Password:
Last login: Fri May  5 01:37:40 EDT 2023 on tty1
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-116-generic x86_64)

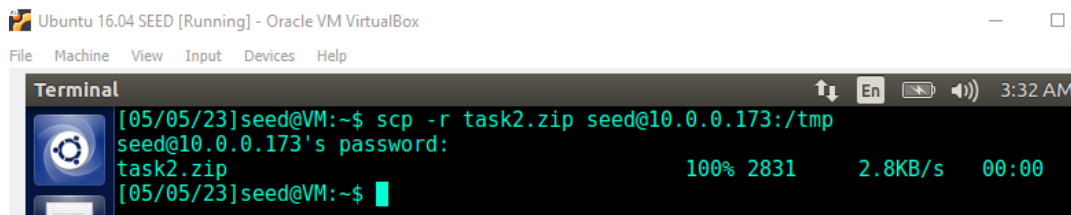
 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage
New release '18.04.6 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

seed@recovery:~$ _

seed@recovery:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:bf:b8:bf
          inet addr:10.0.0.173  Bcast:10.0.0.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:febf:b8bf/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

```

Figure 34: Login to Recovery OS to locate the IP address



```

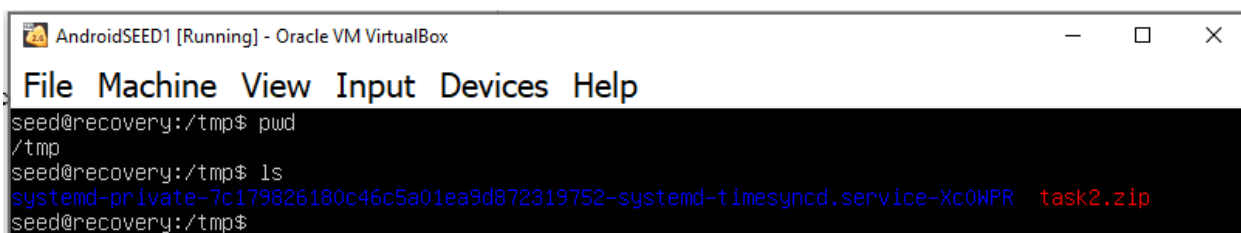
Ubuntu 16.04 SEED [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

Terminal
[05/05/23]seed@VM:~$ scp -r task2.zip seed@10.0.0.173:/tmp
seed@10.0.0.173's password:
task2.zip                                100% 2831      2.8KB/s   00:00
[05/05/23]seed@VM:~$

```

Figure 35: Deliver task2.zip to the Android recovery OS using its IP address

9. In Android VM, change the directory to /tmp and use the ls command to list the files.



```

AndroidSEED1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

seed@recovery:/tmp$ pwd
/tmp
seed@recovery:/tmp$ ls
systemd-private-7c179826180c46c5a01ea9d872319752-systemd-timesyncd.service-Xc0WPR task2.zip
seed@recovery:/tmp$

```

Figure 36: task2.zip in /tmp in Android Recovery

10. Unzip task2.zip in /tmp

```
seed@recovery:/tmp$ unzip task2.zip
Archive:  task2.zip
  creating: task2/
  creating: task2/META-INF/
  creating: task2/META-INF/com/
  creating: task2/META-INF/com/google/
  creating: task2/META-INF/com/google/android/
  inflating: task2/META-INF/com/google/android/update-binary
  inflating: task2/META-INF/com/google/android/my_app_process
seed@recovery:/tmp$
```

Figure 37: unzip task2.zip in /tmp in Android Recovery

11. Navigate to the directory META-INF/com/google/android and locate the update-binary file. Run the file, and if all of the steps were carried out accurately, the Android system should be updated. Subsequently, start the Android OS and verify whether the dummy2 file has been created within the /system directory.

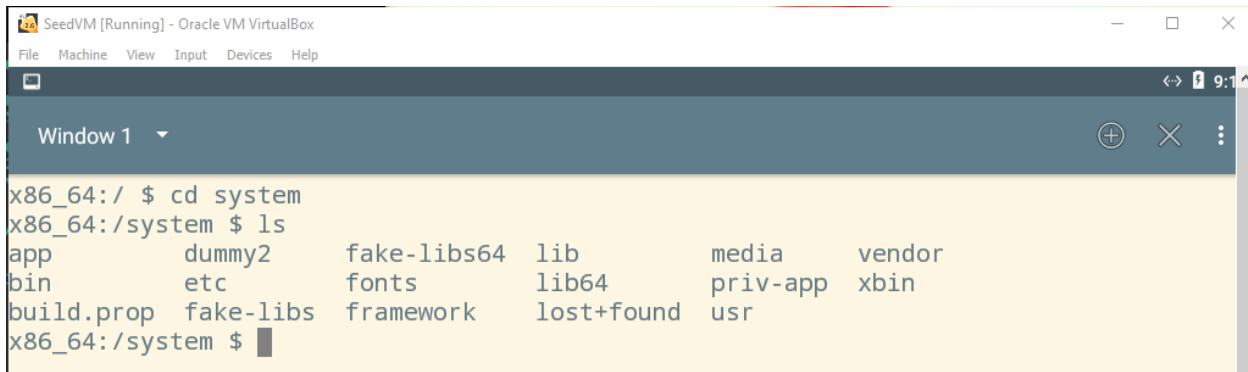
```
seed@recovery:/tmp$ cd task2/META-INF/com/google/android/
seed@recovery:/tmp/task2/META-INF/com/google/android$ ls
my_app_process  update-binary
seed@recovery:/tmp/task2/META-INF/com/google/android$ _
```

Figure 38: ls command to display the files in tmp/task2/META-INF/com/google/android

```
seed@recovery:/tmp/task2/META-INF/com/google/android$ ls -la
total 20
drwxrwxr-x 2 seed seed 4096 May  5 03:07 .
drwxrwxr-x 3 seed seed 4096 May  5 02:10 ..
-rwxr-xr-x 1 seed seed 5116 May  5 02:51 my_app_process
-rwxrwxr-x 1 seed seed  175 May  5 03:07 update-binary
seed@recovery:/tmp/task2/META-INF/com/google/android$ sudo ./update-binary
[sudo] password for seed:
seed@recovery:/tmp/task2/META-INF/com/google/android$ sudo reboot
```

Figure 39: compile update-binary, then reboot

In this lab, we successfully modified the `app_process` program in the Android OS. The `app_process` is executed by the Android runtime during bootstrapping and is responsible for launching the Zygote daemon, which in turn launches all applications. We achieved this by creating a standalone native program using NDK, compiling it and placing it in the appropriate folder structure, and executing it via the `update-binary` file to update the Android system. Upon booting up the Android OS, we verified that the `dummy2` file had been created, thereby demonstrating the successful modification of the app process program.



```

x86_64:/ $ cd system
x86_64:/system $ ls
app          dummy2       fake-libs64  lib          media        vendor
bin          etc          fonts        lib64        priv-app     xbin
build.prop   fake-libs    framework    lost+found   usr
x86_64:/system $

```

Figure 32: dummy2 created in the /system folder, demonstrating root privilege

Lab Task 3: Implement SimpleSU for Getting Root Shell

Another approach to gaining root access on an Android device is to start a root daemon during the booting process, and then use this daemon to help users get a root shell. This method is used by popular rooting OTA packages like SuperSU developed by Chainfire. In this task, we will write a similar daemon to understand how it helps users gain root access. The basic idea is that when a user wants to gain root access, they run a client program that sends a request to the root daemon. Upon receiving the request, the daemon starts a shell process and gives control of the process to the client, allowing the user to control the shell process. The challenging part of this approach is figuring out how to let the user control the shell process created by the daemon. According to The Official Document of Security Updates on Android 4.3, `setuid/setgid` programs have been removed from Android system files and support for filesystem capabilities has been added. This reduces the root attack surface and lowers the likelihood of potential security vulnerabilities.

To complete this task, download the file SimpleSU.zip from https://seedsecuritylabs.org/Labs_20.04/Mobile/Android_Rooting/

Files Needed

- [SimpleSU.zip](#)

1. Boot up SEED Android VM, open an emulator app on the Home screen. Then, open a terminal window.
2. Navigate to the 'system' directory in the file system by typing the command `"cd /system"` in the terminal window.
3. Type the command `ls` to list all files in `/system` directory. Notice that it should not be a `mysfu` file in the `/system` directory.

```

x86_64:/system/xbin $ pwd
/system/xbin
x86_64:/system/xbin $ ls mysu
ls: mysu: No such file or directory
1|x86_64:/system/xbin $

```

Figure 33: mysu is not found in the /system folder

4. In Ubuntu 16.04 SEED VM, create a folder called task3, then under task3, create a subfolder according to the OTA package structure,
META-INF/com/google/android/

```

[05/01/23]seed@VM:~$ mkdir -p task3/META-INF/com/google/android
[05/01/23]seed@VM:~$ ls
android      Documents    lib          source       task2codes   Videos
bin          Downloads    Music        task1         task2.zip
Customization examples.desktop Pictures      task1.zip    task3
Desktop      get-pip.py   Public       task2         Templates
[05/01/23]seed@VM:~$ cd ~/task3/META-INF/com/google/android/
[05/01/23]seed@VM:~/.../android$ gedit update-binary
[05/01/23]seed@VM:~/.../android$

```

Figure 34: META-INF/com/google/android/

5. Type the command "gedit" in the terminal to create the file "update-binary". The code for this file can be found in the Seed Lab manual but we have to modify it. Refer to Figure 35.
6. To enable us to run the file "update-binary", we must add execute permissions. Type the command "chmod a+x update-binary" to achieve this.

```

[05/06/23]seed@VM:~/.../android$ chmod a+x update-binary
[05/06/23]seed@VM:~/.../android$ ls -la
total 12
drwxrwxr-x 2 seed seed 4096 May  6 17:00 .
drwxrwxr-x 3 seed seed 4096 May  6 16:56 ..
-rwxrwxr-x 1 seed seed 134 May  6 17:00 update-binary
[05/06/23]seed@VM:~/.../android$

```

Figure 35: executable update-binary script

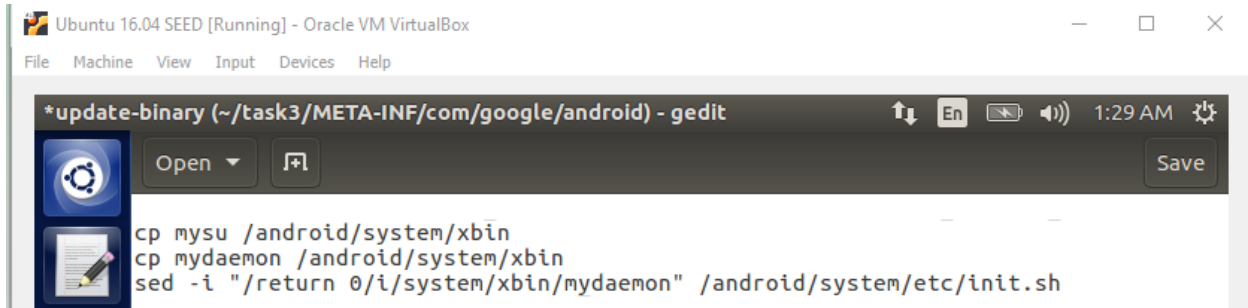


Figure 36: task3 - update-binary contents

7. Create a source code folder named task3codes, then unzip the file SimpleSU.zip from the /Downloads directory to task3codes directory by typing the command, `unzip ~/Downloads/SimpleSu.zip`.

```
[05/07/23]seed@VM:~$ mkdir task3codes
[05/07/23]seed@VM:~$ cd task3codes
[05/07/23]seed@VM:~/task3codes$ unzip ~/Downloads/SimpleSU.zip
Archive:  /home/seed/Downloads/SimpleSU.zip
  creating: SimpleSU/
  creating: SimpleSU/socket_util/
  inflating: SimpleSU/socket_util/socket_util.c
  inflating: SimpleSU/socket_util/socket_util.h
  creating: SimpleSU/mydaemon/
  inflating: SimpleSU/mydaemon/Android.mk
  inflating: SimpleSU/mydaemon/compile.sh
  inflating: SimpleSU/mydaemon/mydaemonsu.c
  inflating: SimpleSU/mydaemon/Application.mk
  inflating: SimpleSU/compile_all.sh
  inflating: SimpleSU/server_loc.h
  creating: SimpleSU/mysu/
  inflating: SimpleSU/mysu/Android.mk
  inflating: SimpleSU/mysu/compile.sh
  inflating: SimpleSU/mysu/mysu.c
  inflating: SimpleSU/mysu/Application.mk
[05/07/23]seed@VM:~/task3codes$
```

Figure 37: unzipping the source code, SimpleSU.zip

8. Change the current directly to SimpleSu by typing `cd SimpleSu`, then list all files in the SimpleSu folder by typing the command `ls`, then type the command `bash compile_all.sh` to execute all the listed commands in this script.

```
[05/01/23]seed@VM:~/task3codes$ cd SimpleSU/
[05/01/23]seed@VM:~/../SimpleSU$ ls
compile_all.sh  mydaemon  mysu  server_loc.h  socket_util
[05/01/23]seed@VM:~/../SimpleSU$ bash compile_all.sh
////////Build Start////////
Compile x86      : mydaemon <= mydaemonsu.c
Compile x86      : mydaemon <= socket_util.c
Executable       : mydaemon
Install          : mydaemon => libs/x86/mydaemon
Compile x86      : mysu <= mysu.c
Compile x86      : mysu <= socket_util.c
Executable       : mysu
Install          : mysu => libs/x86/mysu
////////Build End////////
[05/01/23]seed@VM:~/../SimpleSU$
```

Figure 38: the output of bash compile_all.sh

9. List all the files after compiling `compile_all.sh` by typing the command, we need two output files from `mydaemon` and `mysu`.

```
[05/07/23]seed@VM:~/.../SimpleSU$ ls -la
total 28
drwxr-xr-x 5 seed seed 4096 May 22 2018 .
drwxrwxr-x 3 seed seed 4096 May 7 14:05 ..
-rw-r--r-- 1 seed seed 138 Mar 31 2016 compile_all.sh
drwxr-xr-x 4 seed seed 4096 May 7 14:34 mydaemon
drwxr-xr-x 4 seed seed 4096 May 7 14:34 mysu
-rw-r--r-- 1 seed seed 371 Mar 11 2016 server_loc.h
drwxr-xr-x 2 seed seed 4096 Mar 31 2016 socket_util
[05/07/23]seed@VM:~/.../SimpleSU$
```

Figure 38: the output of bash `compile_all.sh`

10. Change the current directory to `mydaemon/libs/x86` and `mysu/libs/x86` and copy `mydaemon` and `mysu` to `task3/META-INF/com/google/android`. Refer to Figure 39-40.

```
[05/07/23]seed@VM:~/.../SimpleSU$ cd mydaemon/libs/x86
[05/07/23]seed@VM:~/.../x86$ ls
mydaemon
[05/07/23]seed@VM:~/.../x86$ mv mydaemon ~/task3/META-INF/com/google/android/
[05/07/23]seed@VM:~/.../x86$
```

Figure 39: copying `mydaemon` to the required directory, `task3/META-INF/com/google/android`

```
[05/07/23]seed@VM:~/.../x86$ cd ~/task3codes/SimpleSU/mysu/libs/x86
[05/07/23]seed@VM:~/.../x86$ ls
mysu
[05/07/23]seed@VM:~/.../x86$ mv mysu ~/task3/META-INF/com/google/android/
[05/07/23]seed@VM:~/.../x86$
```

Figure 40: copying `mysu` to the required directory, `task3/META-INF/com/google/android`

11. Change the current directory to `task3/META-INF/com/google/android` to verify that we have the three OTA files needed for this task, `mysu`, `mydaemon`, and `update-binary`.

```
[05/07/23]seed@VM:~/.../x86$ cd ~/task3/META-INF/com/google/android/
[05/07/23]seed@VM:~/.../android$ ls
mydaemon mysu update-binary
[05/07/23]seed@VM:~/.../android$ ls -la
total 36
drwxrwxr-x 2 seed seed 4096 May 7 14:41 .
drwxrwxr-x 3 seed seed 4096 May 7 14:01 ..
-rwxr-xr-x 1 seed seed 9232 May 7 14:34 mydaemon
-rwxr-xr-x 1 seed seed 9232 May 7 14:34 mysu
-rwxrwxr-x 1 seed seed 133 May 7 14:01 update-binary
[05/07/23]seed@VM:~/.../android$
```

Figure 41: `mydaemon`, `mysu`, and `update-binary` in `task3/META-INF/com/google/android`

[illegible]

Figure 42: the contents of mydaemon when using the cat command to display

[illegible]

Figure 43: the contents of mysu when using the cat command to display

12. Change the current directory to where we save task3/META-INF/com/google/android so that we can zip the entire folder and transfer the code to the Android recovery OS. To do this, type the command, `cd /home/seed`, then use the `ls` command to verify that we are able to locate task3 folder.

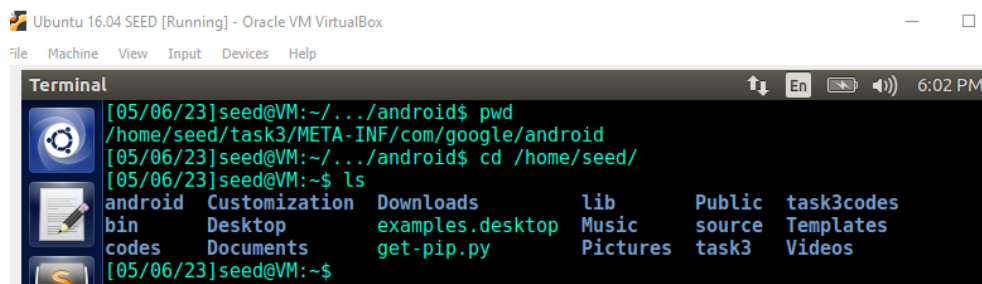


Figure 44: Getting ready to zip task3 folder, cd /home/seed


```
[05/07/23]seed@VM:~$ zip -r task3.zip task3
adding: task3/ (stored 0%)
adding: task3/META-INF/ (stored 0%)
adding: task3/META-INF/com/ (stored 0%)
adding: task3/META-INF/com/google/ (stored 0%)
adding: task3/META-INF/com/google/android/ (stored 0%)
adding: task3/META-INF/com/google/android/update-binary (deflated 41%)
adding: task3/META-INF/com/google/android/mydaemon (deflated 60%)
adding: task3/META-INF/com/google/android/mysu (deflated 66%)
[05/07/23]seed@VM:~$
```

Figure 45: Zipping task3 folder

13. We have successfully zip task3.zip in /home/seed, then we are going to transfer the task3.zip folder to the Android Recovery OS.

```
[05/07/23]seed@VM:~$ ls
android      Documents    lib          source       Templates
bin          Downloads    Music        task3        Videos
Customization examples.desktop Pictures      task3codes
Desktop      get-pip.py   Public       task3.zip
```

Figure 46: task3.zip

14. To locate the IP address of the AndroidSEED VM, login to Android recovery OS, then type `ifconfig` command to obtain the required IP address. Power off the Android VM if it is still running a normal mode, then restart the VM, and quickly hold down the shift key right before you see the Virtual box logo booting up the VM to enter the GNU mode and select Ubuntu so that we can enter into Android recovery mode.

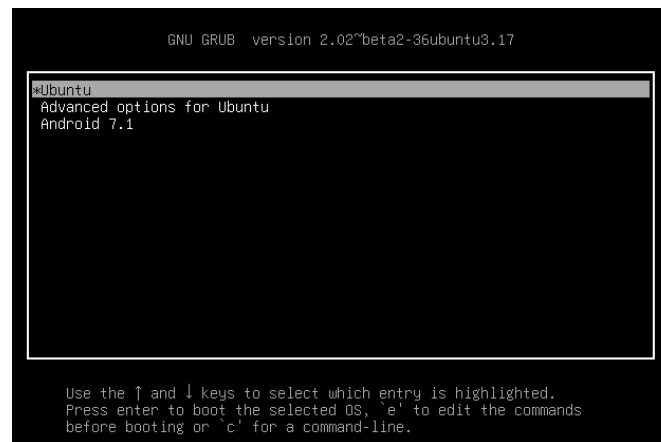
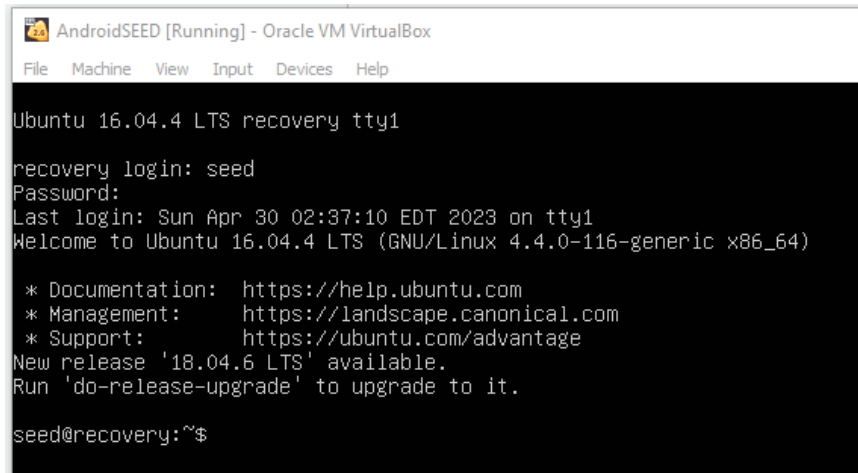


Figure 47: GNU mode used for Android Recovery Mode

15. When prompted for a recovery login, enter “seed” for the username and “dees” for the password.



```

AndroidSEED [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

Ubuntu 16.04.4 LTS recovery tty1

recovery login: seed
Password:
Last login: Sun Apr 30 02:37:10 EDT 2023 on tty1
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-116-generic x86_64)

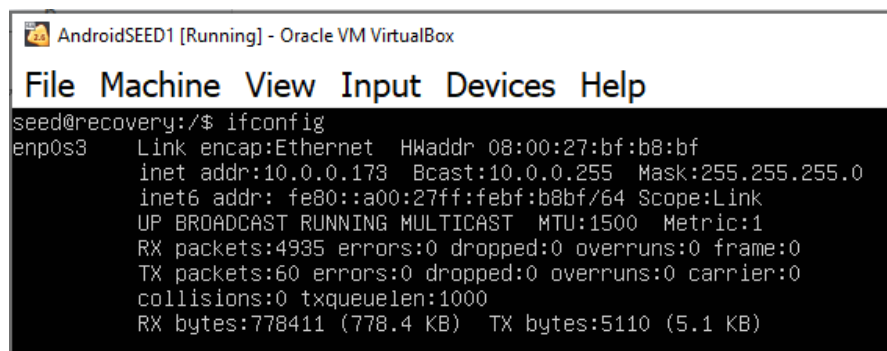
 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage
New release '18.04.6 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

seed@recovery:~$

```

Figure 48: Android Recovery Mode

16. Type the command `ifconfig`, and locate the IP address, in this case, it is 10.0.0.173.



```

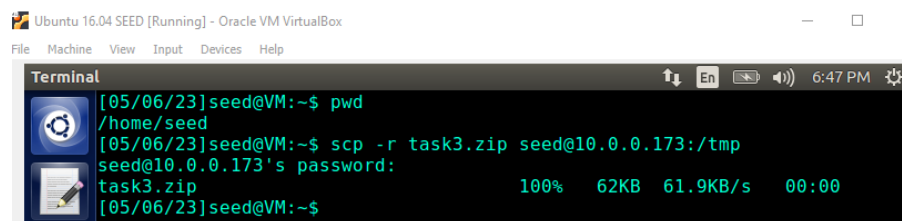
AndroidSEED1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

seed@recovery:/$ ifconfig
enp0s3  Link encap:Ethernet  HWaddr 08:00:27:bf:b8:bf
        inet addr:10.0.0.173  Bcast:10.0.0.255  Mask:255.255.255.0
        inet6 addr: fe80::a00:27ff:febf:b8bf/64  Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:4935 errors:0 dropped:0 overruns:0 frame:0
        TX packets:60 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:778411 (778.4 KB)  TX bytes:5110 (5.1 KB)

```

Figure 49: Locating the IP address of the AndroidSEED VM, 10.0.0.173.

17. Return to Ubuntu16.04 SEEDVM and type the command, `scp -r task3.zip seed@10.0.0.173:/tmp` to transfer task3.zip to the AndroidSEED VM.



```

Ubuntu 16.04 SEED [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

Terminal
[05/06/23]seed@VM:~$ pwd
/home/seed
[05/06/23]seed@VM:~$ scp -r task3.zip seed@10.0.0.173:/tmp
seed@10.0.0.173's password:
task3.zip                                100% 62KB 61.9KB/s 00:00
[05/06/23]seed@VM:~$

```

Figure 50: transferring task3.zip to the AndroidSEED VM

18. Switch to the AndroidSEED VM, then change the current directory to /tmp, then use the command `ls` to list all the files. Refer to Figure 50 - task3.zip has been successfully transferred to the AndroidSEED VM or the Android Recovery OS.

```

AndroidSEED1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
seed@recovery:/$ cd /tmp
seed@recovery:/tmp$ ls
systemd-private-8a682313efc347dda1d06b338323f79-systemd-time
seed@recovery:/tmp$ ls -la
total 92
drwxrwxrwt  8 root root  4096 May  6 18:18 .
drwxr-xr-x 23 root root  4096 Mar 26 2018 ..
drwxrwxrwt  2 root root  4096 May  6 16:46 .font-unix
drwxrwxrwt  2 root root  4096 May  6 16:46 .ICE-unix
drwx----- 3 root root  4096 May  6 17:35 systemd-private-8a
timesyncd.service-U5e7M1
-rw-rw-r--  1 seed seed 61426 May  6 18:18 task3.zip
drwxrwxrwt  2 root root  4096 May  6 16:46 .Test-unix
drwxrwxrwt  2 root root  4096 May  6 16:46 .X11-unix
drwxrwxrwt  2 root root  4096 May  6 16:46 .XIM-unix
seed@recovery:/tmp$

```

Figure 51: task3.zip in the /tmp folder in Android root

19. Unzip the task3.zip.

```

AndroidSEED3 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
seed@recovery:/tmp$ unzip task3.zip
Archive:  task3.zip
  creating: task3/
  creating: task3/x86/
  creating: task3/META-INF/
  creating: task3/META-INF/com/
  creating: task3/META-INF/com/google/
  creating: task3/META-INF/com/google/android/
  inflating: task3/META-INF/com/google/android/update-binary
  inflating: task3/META-INF/com/google/android/mydaemon
  inflating: task3/META-INF/com/google/android/mysu
seed@recovery:/tmp$

```

Figure 52: Unzipping task3.zip in the /tmp folder in Android root

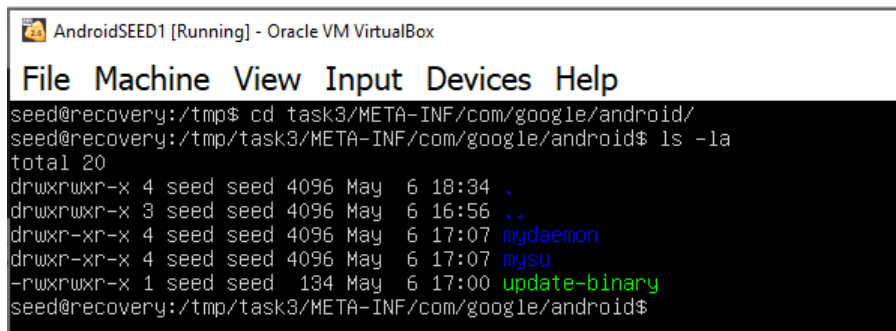
```

AndroidSEED1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
seed@recovery:/tmp$ ls -la
total 96
drwxrwxrwt  9 root root  4096 May  6 18:23 .
drwxr-xr-x 23 root root  4096 Mar 26 2018 ..
drwxrwxrwt  2 root root  4096 May  6 16:46 .font-unix
drwxrwxrwt  2 root root  4096 May  6 16:46 .ICE-unix
drwx----- 3 root root  4096 May  6 17:35 systemd-private-8a682
timesyncd.service-U5e7M1
drwxrwxr-x  4 seed seed  4096 May  6 17:15 task3
-rw-rw-r--  1 seed seed 61426 May  6 18:18 task3.zip
drwxrwxrwt  2 root root  4096 May  6 16:46 .Test-unix
drwxrwxrwt  2 root root  4096 May  6 16:46 .X11-unix
drwxrwxrwt  2 root root  4096 May  6 16:46 .XIM-unix
seed@recovery:/tmp$ ls -la task3
total 16
drwxrwxr-x  4 seed seed  4096 May  6 17:15 .
drwxrwxrwt  9 root root  4096 May  6 18:23 ..
drwxrwxr-x  3 seed seed  4096 May  6 16:56 META-INF
drwxrwxr-x  4 seed seed  4096 May  6 17:40 x86
seed@recovery:/tmp$

```

Figure 53: Unzipped task3.zip in the /tmp folder in Android root

20. Change the current directory to `/task3/META-INF/com/google/android` and type the command `ls` to list all files.



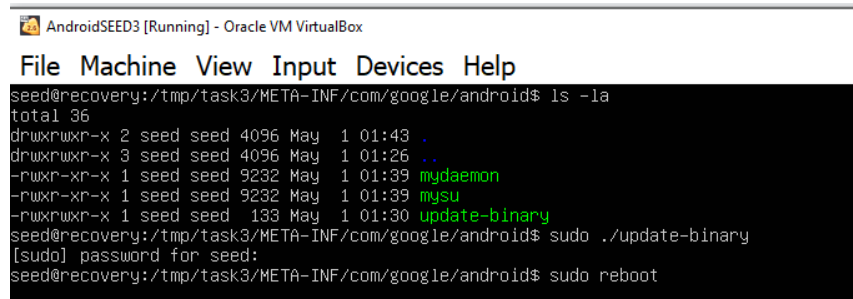
```

AndroidSEED1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
seed@recovery:/tmp$ cd task3/META-INF/com/google/android/
seed@recovery:/tmp/task3/META-INF/com/google/android$ ls -la
total 20
drwxrwxr-x 4 seed seed 4096 May  6 18:34 .
drwxrwxr-x 3 seed seed 4096 May  6 16:56 ..
drwxr-xr-x 4 seed seed 4096 May  6 17:07 mydaemon
drwxr-xr-x 4 seed seed 4096 May  6 17:07 mysu
-rwxrwxr-x 1 seed seed 134 May  6 17:00 update-binary
seed@recovery:/tmp/task3/META-INF/com/google/android$

```

Figure 54: mydaemon, mysu, and update-binary in `/task3/META-INF/com/google/android`

21. Execute the command `sudo ./update-binary` then, enter the password “dees and `sudo reboot`.



```

AndroidSEED3 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
seed@recovery:/tmp/task3/META-INF/com/google/android$ ls -la
total 36
drwxrwxr-x 2 seed seed 4096 May  1 01:43 .
drwxrwxr-x 3 seed seed 4096 May  1 01:26 ..
-rwxr-xr-x 1 seed seed 9232 May  1 01:39 mydaemon
-rwxr-xr-x 1 seed seed 9232 May  1 01:39 mysu
-rwxrwxr-x 1 seed seed 133 May  1 01:30 update-binary
seed@recovery:/tmp/task3/META-INF/com/google/android$ sudo ./update-binary
[sudo] password for seed:
seed@recovery:/tmp/task3/META-INF/com/google/android$ sudo reboot

```

Figure 55: execute scripts in the file, update-binary

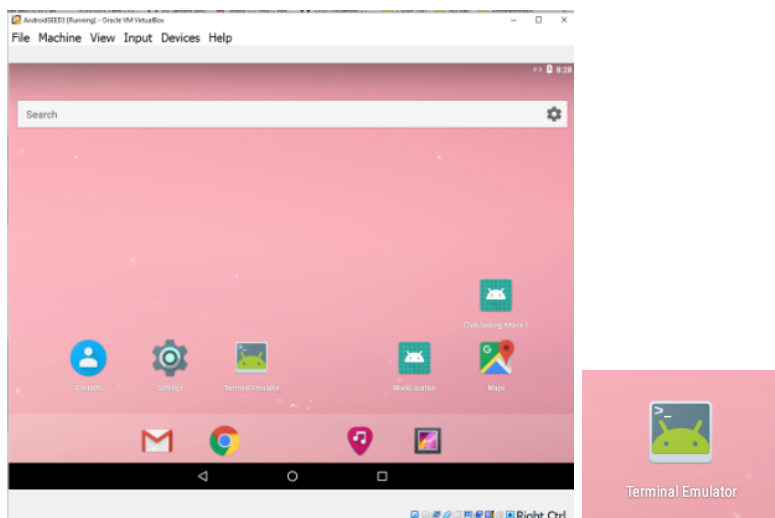
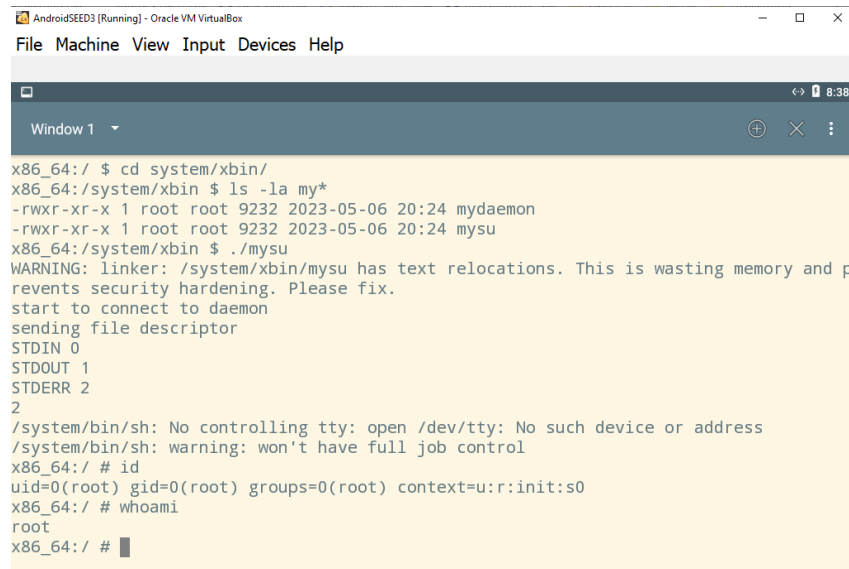


Figure 56: After executing `sudo reboot`, the VM is booting up into Android GUI.

22. Open the Terminal Emulator app, and type `cd system/xbin`, then `ls -la my*` to display any files that start with my. Notice that mydaemon and mysu are listed below, refer to Figure



```

x86_64:/ $ cd system/xbin/
x86_64:/system/xbin $ ls -la my*
-rwxr-xr-x 1 root root 9232 2023-05-06 20:24 mydaemon
-rwxr-xr-x 1 root root 9232 2023-05-06 20:24 mysu
x86_64:/system/xbin $ ./mysu
WARNING: linker: /system/xbin/mysu has text relocations. This is wasting memory and p
revents security hardening. Please fix.
start to connect to daemon
sending file descriptor
STDIN 0
STDOUT 1
STDERR 2
2
/system/bin/sh: No controlling tty: open /dev/tty: No such device or address
/system/bin/sh: warning: won't have full job control
x86_64:/ # id
uid=0(root) gid=0(root) groups=0(root) context=u:r:init:s0
x86_64:/ # whoami
root
x86_64:/ #

```

Figure 57: mysu and daemon in /system/xbin gaining root access

In task3, we were able to start a root daemon during the booting process, and then use this daemon to get a root shell. We achieve this by placing mysu and mydaemon in the root directory. To gain root privileges, we had to run a client program that sent a request to the root daemon, which then started a shell process and gave it to the client and finally, we have the root access.

There are several ways to defend against Android Rooting Device attacks:

1. Lock the bootloader
2. Keep the device's software up-to-date with the latest security
3. Disable USB debugging not allowing a device to connect to a computer and run commands on it.
4. Use strong passwords for unlocking the device to prevent unauthorized access to the device.
5. Avoid sideloading apps from unknown sources, only download apps from the official app