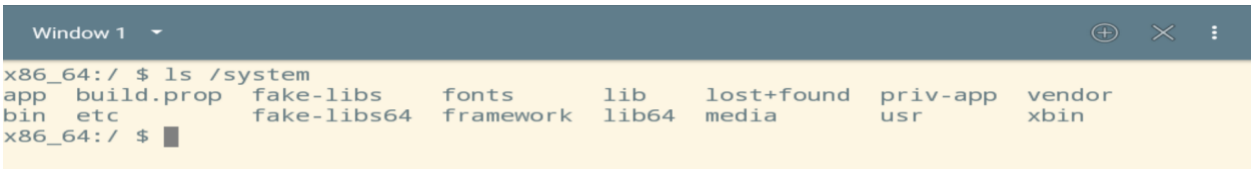
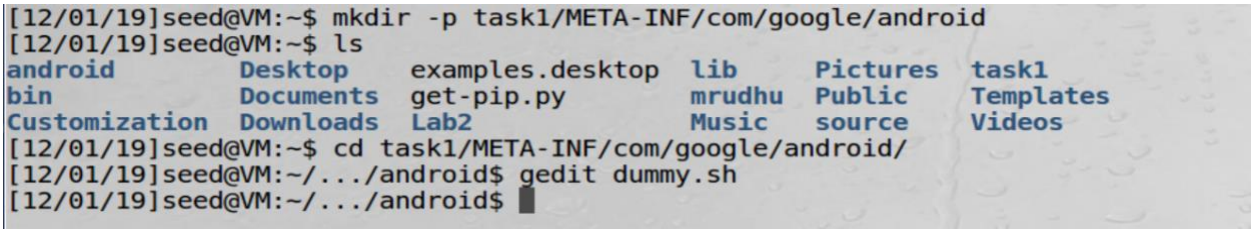
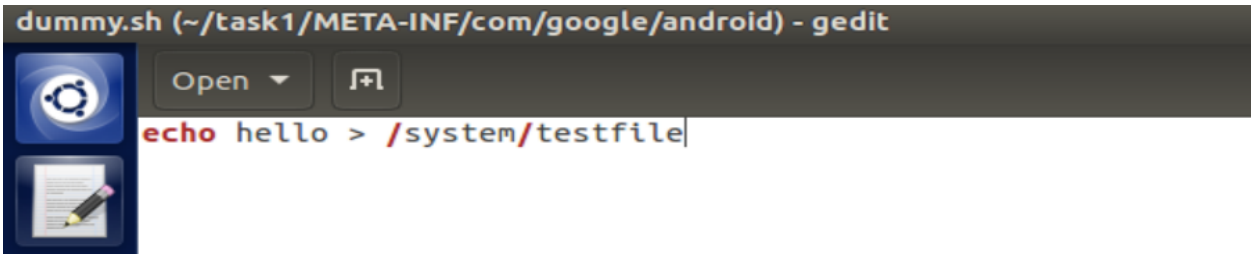
LAB 13: ANDROID ROOTING LAB

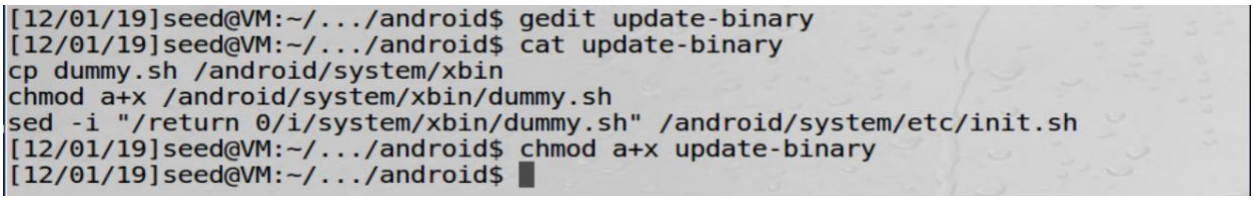
Task 1: Build a simple OTA package

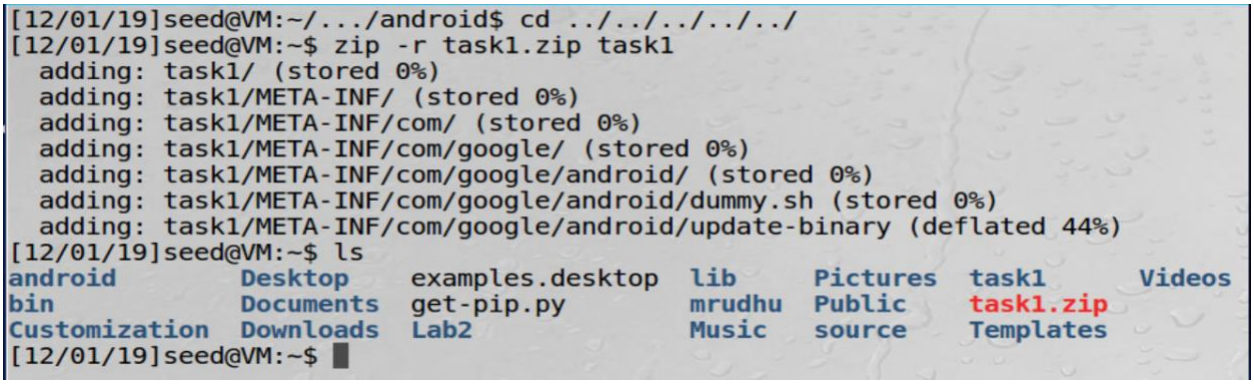


The above screenshot is a list of files present in the android system folder.



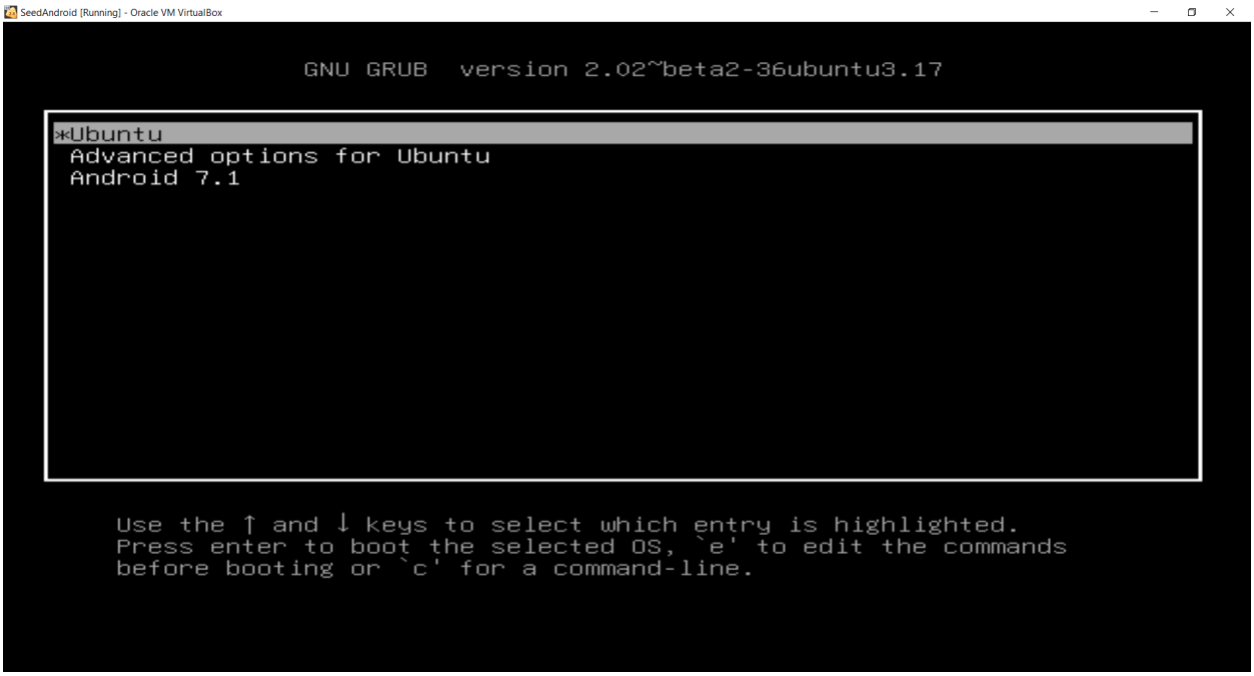




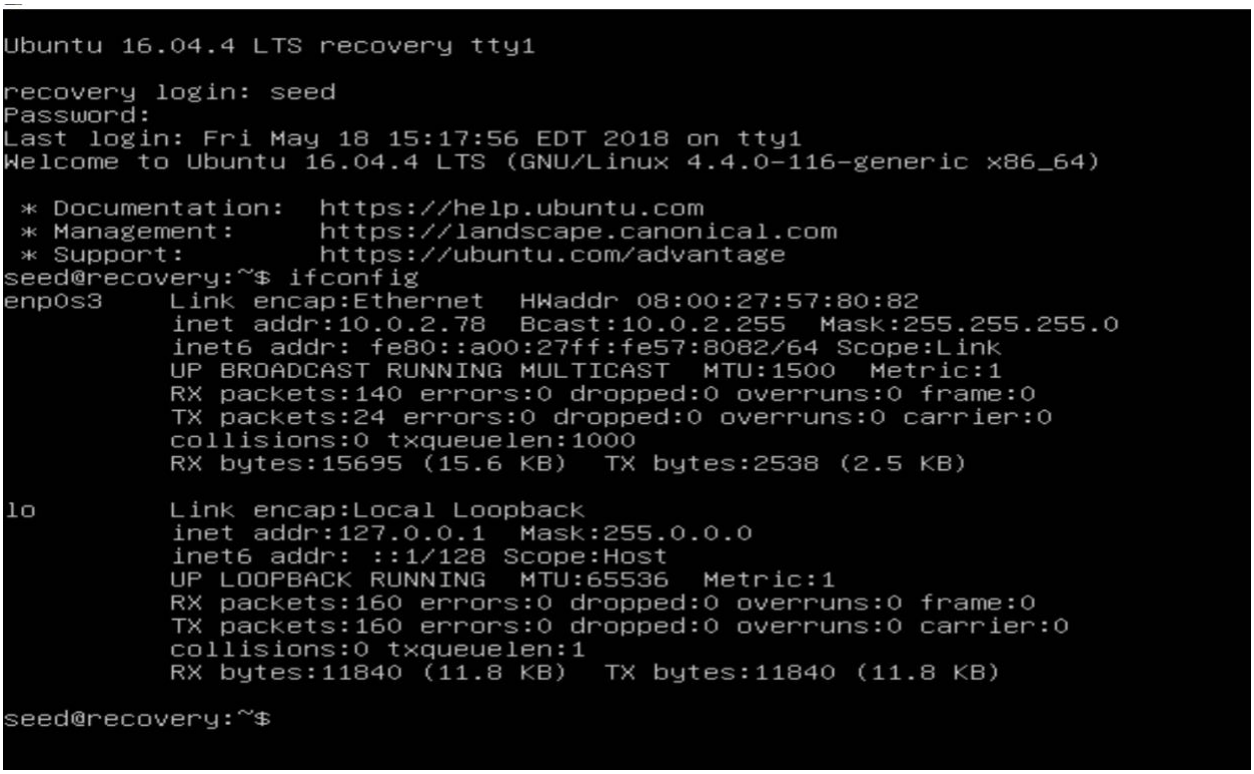


The above screenshots show that we have created the required folder structure so that we add the update binary file in the required android folder. We create a dummy file in the android

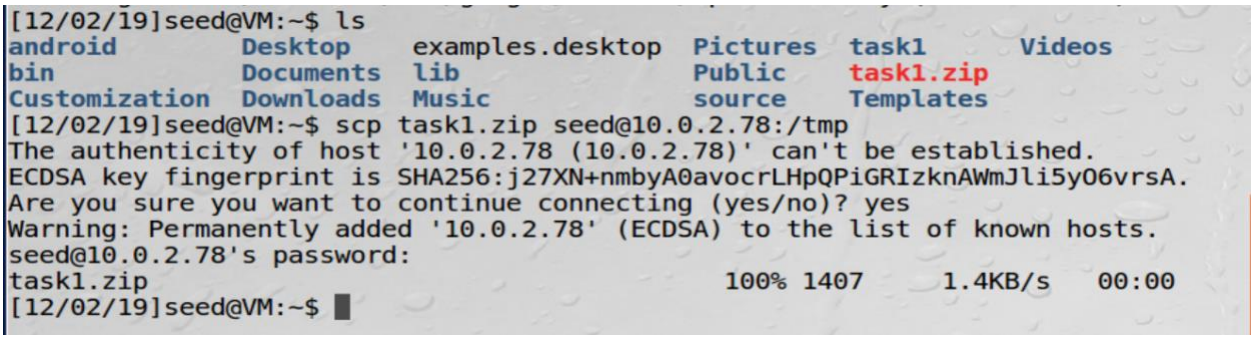
folder. We give the update-binary file executable permissions. We then create a zip file of the entire package.



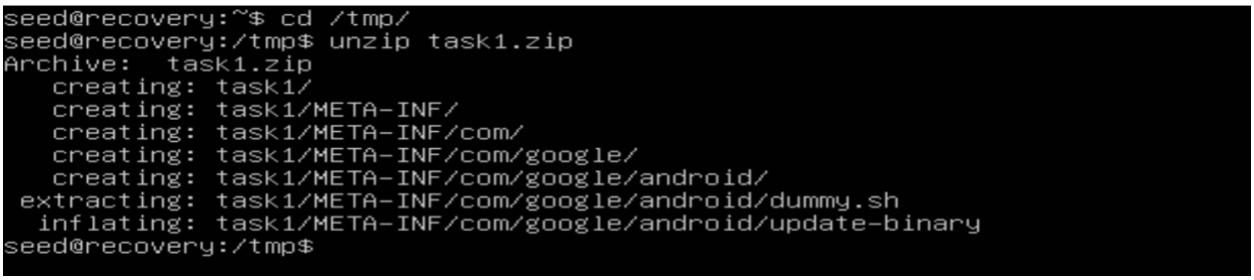
On SeedAndroid VM we click left+shift when we see the virtual box sign due to which we obtain 3 options. To get into recovery OS we select Ubuntu from the given 3 options.

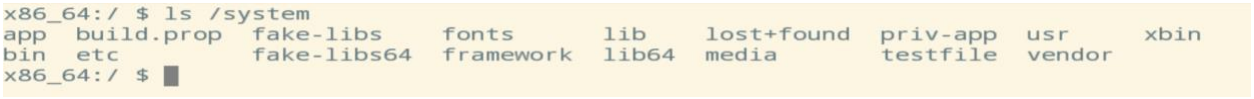


We login to the recovery OS and find the IP address of the android VM.



We send the zip package from the SeedUbuntu VM to the recovery OS and place it in the /tmp folder of the recovery OS.





On restarting the Android OS, we can find the dummy folder in /system/. When Android kernel is loaded init.sh is executed which is modified and hence dummy.sh is executed which creates a folder in /system.

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. Hence, this runs with root privileges. Now that this is running with root privileges, this will create a file called dummy in the system folder which requires root permissions. Normally 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 and checking the system folder contents.

Task 2: Inject code via app-process

```
[12/02/19]seed@VM:~$ gedit app_process.c
[12/02/19]seed@VM:~$ cat 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;
}

[12/02/19]seed@VM:~$
```

```
[12/02/19]seed@VM:~$ gedit Application.mk
[12/02/19]seed@VM:~$ cat Application.mk
APP_ABI := x86
APP_PLATFORM := android-21
APP_STL := stlport_static
APP_BUILD_SCRIPT := Android.mk
[12/02/19]seed@VM:~$
```

```
[12/02/19]seed@VM:~$ gedit Android.mk
[12/02/19]seed@VM:~$ cat Android.mk
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := app_process
LOCAL_SRC_FILES := app_process.c
include $(BUILD_EXECUTABLE
[12/02/19]seed@VM:~$
```

```
[12/02/19]seed@VM:~$ ls
android          Desktop          Lab2             Pictures   task2
Android.mk       Documents        lib              Public     task2_code
Application.mk   Downloads        mrudhu           source     Templates
bin              examples.desktop Music            task1      Videos
Customization    get-pip.py       my_app_process.c task1.zip
[12/02/19]seed@VM:~$
```

```
[12/02/19]seed@VM:~/task2_code$ ls
Android.mk  Application.mk  my_app_process.c
[12/02/19]seed@VM:~/task2_code$
```

```
[12/02/19]seed@VM:~/task2_code$ export NDK_PROJECT_PATH=.
[12/02/19]seed@VM:~/task2_code$ ndk-build NDK_APPLICATION_MK=./Application.mk
Compile x86    : my_app_process <= my_app_process.c
Executable     : my_app_process
Install        : my_app_process => libs/x86/my_app_process
[12/02/19]seed@VM:~/task2_code$ ls
Android.mk  Application.mk  libs  my_app_process.c  obj
[12/02/19]seed@VM:~/task2_code$
```

```
[12/02/19]seed@VM:~$ mkdir -p task2/META-INF/com/google/android
[12/02/19]seed@VM:~$ cd task2/META-INF/com/google/android/
[12/02/19]seed@VM:~/.../android$ gedit update-binary
[12/02/19]seed@VM:~/.../android$ cat update-binary
mv /android/system/bin/app_process64 /android/system/bin/app_process_original
cp app_process /android/system/bin/app_process64
chmod a+x /android/system/bin/app_process64
[12/02/19]seed@VM:~/.../android$ ls
app_process  update-binary
[12/02/19]seed@VM:~/.../android$
```

```
[12/02/19]seed@VM:~/.../android$ chmod a+x update-binary
[12/02/19]seed@VM:~/.../android$ cd ../../../../../
[12/02/19]seed@VM:~$ ls
android          Customization    get-pip.py  source     Templates
Android.mk       Desktop          lib         task1      Videos
Application.mk   Documents        Music       task1.zip
app_process.c    Downloads        Pictures    task2
bin              examples.desktop Public      task2_code
[12/02/19]seed@VM:~$
```

```
[12/02/19]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 59%)
  adding: task2/META-INF/com/google/android/app_process (deflated 72%)
[12/02/19]seed@VM:~$ ls
android          Customization    get-pip.py  source     task2.zip
Android.mk       Desktop          lib         task1      Templates
Application.mk   Documents        Music       task1.zip  Videos
app_process.c    Downloads        Pictures    task2
bin              examples.desktop Public      task2_code
[12/02/19]seed@VM:~$
```

```
[12/02/19]seed@VM:~$ scp task2.zip seed@10.0.2.78:/tmp
seed@10.0.2.78's password:
task2.zip                                     100% 2821     2.8KB/s   00:00
[12/02/19]seed@VM:~$
```

We create the appropriate folder structure and create the update-binary file. We give the update-binary file executable permissions. We compile the my_app_process file using NDK which creates the file in x86 folder and we place it in the android folder. We then zip the entire package.

```
seed@recovery:~$ cd /tmp/
seed@recovery:/tmp$ ls
systemd-private-c68322b12ebf40b7ac6c2f3591c1e950-systemd-timesyncd.service-dSYVJb  task2.zip
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/app_process
seed@recovery:/tmp$ ls
systemd-private-c68322b12ebf40b7ac6c2f3591c1e950-systemd-timesyncd.service-dSYVJb  task2  task2.zip
seed@recovery:/tmp$ _
```

```
seed@recovery:/tmp$ cd /tmp/task2/META-INF/com/google/android/
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$ ll
total 20
drwxrwxr-x 2 seed seed 4096 Dec  2 16:03 ./
drwxrwxr-x 3 seed seed 4096 Dec  2 15:56 ../
-rwxr-xr-x 1 seed seed 5116 Dec  2 16:03 app_process*
-rwxrwxr-x 1 seed seed  171 Dec  2 16:00 update-binary*
seed@recovery:/tmp/task2/META-INF/com/google/android$ sudo reboot
```

We test whether there exists a connection to the Android VM and we send the package from the SeedUbuntu VM to the recovery OS in Android VM. We extract the package in the recovery OS and run the update-binary script.

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

The above screenshot shows that dummy2 file is created in system folder and the attack is successful.

When Android starts, it always runs a program called app_process after init using root privilege. This app_process starts the zygote daemon whose work is to start an application, and this is the parent of all app processes. Hence, we modify the app_process and it will launch something of our choice along with launching the zygote process. We create the OTA package by creating the update-binary in the required folder hierarchy. The update binary file will rename the app_process64 file into something else say 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_process64. The file we created is compiled in such a way that it can run on any system. The app_process64 we created will internally call the original app_process64 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.

Task 3: Implement SimpleSU for Getting Root Shell

```
[12/02/19]seed@VM:~$ wget https://seedsecuritylabs.org/Labs_16.04/Mobile/Android
_Rooting/SimpleSU.zip
--2019-12-02 16:21:15--  https://seedsecuritylabs.org/Labs_16.04/Mobile/Android_
Rooting/SimpleSU.zip
Resolving seedsecuritylabs.org (seedsecuritylabs.org)... 185.199.110.153, 185.19
9.111.153, 185.199.109.153, ...
Connecting to seedsecuritylabs.org (seedsecuritylabs.org)|185.199.110.153|:443..
. connected.
HTTP request sent, awaiting response... 200 OK
Length: 11419 (11K) [application/zip]
Saving to: 'SimpleSU.zip'

SimpleSU.zip        100%[===================>]  11.15K  --.-KB/s    in 0s

2019-12-02 16:21:16 (36.0 MB/s) - 'SimpleSU.zip' saved [11419/11419]

[12/02/19]seed@VM:~$ ls
android         Customization     get-pip.py  SimpleSU.zip  task2_code
Android.mk      Desktop           lib         source        task2.zip
Application.mk  Documents         Music       task1         Templates
app_process.c   Downloads         Pictures    task1.zip     Videos
bin             examples.desktop  Public      task2
[12/02/19]seed@VM:~$
```

```
[12/02/19]seed@VM:~$ unzip SimpleSU.zip
Archive:  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
[12/02/19]seed@VM:~$ ls
android         Customization     get-pip.py  SimpleSU      task2
Android.mk      Desktop           lib         SimpleSU.zip  task2_code
Application.mk  Documents         Music       source        task2.zip
app_process.c   Downloads         Pictures    task1         Templates
bin             examples.desktop  Public      task1.zip     Videos
[12/02/19]seed@VM:~$
```

```
[12/02/19]seed@VM:~$ cd SimpleSU/
[12/02/19]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////////////
[12/02/19]seed@VM:~/SimpleSU$
```

```
[12/02/19]seed@VM:~/SimpleSU$ ls
compile_all.sh  mydaemon  mysu  server_loc.h  socket_util
[12/02/19]seed@VM:~/SimpleSU$ cd mydaemon/libs/x86/
[12/02/19]seed@VM:~/.../x86$ ls
mydaemon
[12/02/19]seed@VM:~/.../x86$
```

```
[12/02/19]seed@VM:~/SimpleSU$ cd mysu/libs/x86/
[12/02/19]seed@VM:~/.../x86$ ls
mysu
[12/02/19]seed@VM:~/.../x86$
```

We unzip the simpleSU package. We then give executable permissions to compile_all.sh file and run the file.

```
[12/02/19]seed@VM:~$ ls
android         Customization     get-pip.py  SimpleSU      task2
Android.mk      Desktop           lib         SimpleSU.zip  task2_code
Application.mk  Documents         Music       source        task2.zip
app_process.c   Downloads         Pictures    task1         Templates
bin             examples.desktop  Public      task1.zip     Videos
[12/02/19]seed@VM:~$ mkdir -p task3/META-INF/com/google/android
[12/02/19]seed@VM:~$ ls
android         Customization     get-pip.py  SimpleSU      task2       Videos
Android.mk      Desktop           lib         SimpleSU.zip  task2_code
Application.mk  Documents         Music       source        task2.zip
app_process.c   Downloads         Pictures    task1         task3
bin             examples.desktop  Public      task1.zip     Templates
[12/02/19]seed@VM:~$ cd task3/
[12/02/19]seed@VM:~/task3$ mkdir x86
[12/02/19]seed@VM:~/task3$ ls -l
total 8
drwxrwxr-x 3 seed seed 4096 Dec  2 16:26 META-INF
drwxrwxr-x 2 seed seed 4096 Dec  2 16:26 x86
[12/02/19]seed@VM:~/task3$
```

Then we copy the mysu and mydaemon files to the android folder

```
[12/02/19]seed@VM:~/task3$ cd x86/
[12/02/19]seed@VM:~/.../x86$ ls
mydaemon  mysu
[12/02/19]seed@VM:~/.../x86$ ls -l
total 24
-rwxr-xr-x 1 seed seed 9232 Dec  2 16:34 mydaemon
-rwxr-xr-x 1 seed seed 9232 Dec  2 16:35 mysu
[12/02/19]seed@VM:~/.../x86$
```

```
[12/02/19]seed@VM:~/task3$ cd META-INF/com/google/android/
[12/02/19]seed@VM:~/.../android$ gedit update-binary
[12/02/19]seed@VM:~/.../android$ cat update-binary
mv /android/system/bin/app_process64 /android/system/bin/app_process_original
cp ../../../../x86/mydaemon /android/system/bin/app_process64
cp ../../../../x86/mysu /android/system/xbin/mysu
chmod a+x /android/system/bin/app_process64
chmod a+x /android/system/xbin/mysu
[12/02/19]seed@VM:~/.../android$
```

```
[12/02/19]seed@VM:~/.../android$ chmod a+x update-binary
[12/02/19]seed@VM:~/.../android$ cd ../../../../../
[12/02/19]seed@VM:~$ ls
android          Customization     get-pip.py  SimpleSU      task2       Videos
Android.mk       Desktop           lib         SimpleSU.zip  task2_code
Application.mk   Documents         Music       source        task2.zip
app_process.c    Downloads         Pictures    task1         task3
bin              examples.desktop  Public      task1.zip     Templates
[12/02/19]seed@VM:~$ zip -r task3.zip task3
  adding: task3/ (stored 0%)
  adding: task3/x86/ (stored 0%)
  adding: task3/x86/mydaemon (deflated 60%)
  adding: task3/x86/mysu (deflated 66%)
  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 63%)
[12/02/19]seed@VM:~$ ls
android          Desktop           Music       task1       task3.zip
Android.mk       Documents         Pictures    task1.zip   Templates
Application.mk   Downloads         Public      task2       Videos
app_process.c    examples.desktop  SimpleSU    task2_code
bin              get-pip.py        SimpleSU.zip  task2.zip
Customization    lib               source      task3
[12/02/19]seed@VM:~$
```

```
[12/02/19]seed@VM:~$ scp task3.zip seed@10.0.2.78:/tmp
seed@10.0.2.78's password:
task3.zip                                     100% 8541     8.3KB/s   00:00
[12/02/19]seed@VM:~$
```

The entire OTA package is zipped and copied to /tmp folder in Recovery OS using scp. Run the update-binary in the required folder of Recover OS. Reboot Android OS and run client program mysu in /system/xbin folder. We will a root shell. Hence, we can run any command using root privileges.

```
seed@recovery:~$ cd /tmp/
seed@recovery:/tmp$ ls
systemd-private-b26bb58451f9414bafed77288de2ab39-systemd-timesyncd.service-tj9CUW  task3.zip
seed@recovery:/tmp$ unzip task3.zip
Archive:  task3.zip
   creating: task3/
   creating: task3/x86/
  inflating: task3/x86/mydaemon
  inflating: task3/x86/mysu
   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
seed@recovery:/tmp$ ls
systemd-private-b26bb58451f9414bafed77288de2ab39-systemd-timesyncd.service-tj9CUW  task3  task3.zip
seed@recovery:/tmp$ _
```

```
seed@recovery:/tmp$ cd task3/META-INF/com/google/android/
seed@recovery:/tmp/task3/META-INF/com/google/android$ ls
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$ ll
total 12
drwxrwxr-x 2 seed seed 4096 Dec  2 16:45 ./
drwxrwxr-x 3 seed seed 4096 Dec  2 16:26 ../
-rwxrwxr-x 1 seed seed  270 Dec  2 16:45 update-binary*
seed@recovery:/tmp/task3/META-INF/com/google/android$ sudo reboot
```

```
x86_64:/ $ id
uid=10036(u0_a36) gid=10036(u0_a36) groups=10036(u0_a36),3003(inet),9997(everybody),5
0036(all_a36) context=u:r:untrusted_app:s0:c512,c768
x86_64:/ $ 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:/ #
```

```
x86_64:/ # whoami
root
x86_64:/ # ps | grep mysu
u0_a36    3101  3092  5064   1864           0 0000000000 S mysu
x86_64:/ # ls /proc/310
3101/  3102/
x86_64:/ # ls /proc/3101/fd/
0  1  2  3
x86_64:/ # ls /proc/3092/fd/
0  1  10  2
x86_64:/ # ls /proc/3102/fd/
0  1  10  2  4  5  6  7  9
x86_64:/ #
```

We execute the mysu file, we get root shell.

Here we want to start a root daemon so that we get a root shell and execute any command of our choice. 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. Hence, 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. Therefore, the user now has complete control of the shell process.

Q&A

1) Server launches the original app process binary

```
246  int main(int argc, char** argv) {
247      pid_t pid = fork();
248      if (pid == 0) {
249          //initialize the daemon if not running
250          if (!detect_daemon())
251              run_daemon(argv);
252          }
253      else {
254          argv[0] = APP_PROCESS;
255          execve(argv[0], argv, environ);
256      }
257  }
258
```

Filename: mydaemonsu.c
Function: main() Line:255

2) Client sends its FDs

```
112      send_fd(socket, STDIN_FILENO);      //STDIN_FILENO = 0
113      send_fd(socket, STDOUT_FILENO);     //STDOUT_FILENO = 1
114      send_fd(socket, STDERR_FILENO);     //STDERR_FILENO = 2
115
```

Filename: mysu.c
Function: connect_daemon() Line:112, 113, 114

3) Server forks to a child process

```
245
246  int main(int argc, char** argv) {
247      pid_t pid = fork();
248      if (pid == 0) {
249          //initialize the daemon if not running
250          if (!detect_daemon())
251              run_daemon(argv);
252          }
```

Filename: mydaemonsu.c
Function: main() Line:247

4) Child process receives client's FDs

```
143  int child_process(int socket, char** argv){
144      //handshake
145      handshake_server(socket);
146
147      int client_in = recv_fd(socket);
148      int client_out = recv_fd(socket);
149      int client_err = recv_fd(socket);
150
```

Filename: mydaemonsu.c
Function: child_process() Line: 147, 148, 149

5) Child process redirects its standard I/O FDs

```
151
152      dup2(client_in, STDIN_FILENO);      //STDIN_FILENO = 0
153      dup2(client_out, STDOUT_FILENO);    //STDOUT_FILENO = 1
154      dup2(client_err, STDERR_FILENO);    //STDERR_FILENO = 2
155
```

Filename: mydaemonsu.c
Function: child_process() Line: 152, 153, 154

6) Child process launches a root shell

```
150      }
151      //if root
152      //launch default shell directly
153      char* shell[] = {"/system/bin/sh", NULL};
154      execve(shell[0], shell, NULL);
155      return (EXIT_SUCCESS);
156  }
157
158
```

Filename: mysu.c
Function: main() Line: 154