

Computer Security

Lab 12 Report

Android Rooting

Chirag Sachdev

680231131

Task 1:

Build simple OTA package

```
Window 1
x86_64:/ $ ls /system/
app  build.prop  fake-libs  fonts  lib  lost+found  priv-app  vendor
bin  etc        fake-libs64  framework  lib64  media  usr  xbin
x86_64:/ $ echo hello > /system/dummy
/system/bin/sh: can't create /system/dummy: Permission denied
x86_64:/ $
```

Here we see that we cannot create a dummy file in the system/ dir of the android OS.

```
[12/06/2019]Chirag@VM:~/.../Lab12-Rooting$ lr -l Task1/
Task1/:
total 4
drwxrwxr-x 3 seed seed 4096 Dec  6 19:51 META_INF

Task1/META_INF:
total 4
drwxrwxr-x 3 seed seed 4096 Dec  6 19:51 com

Task1/META_INF/com:
total 4
drwxrwxr-x 3 seed seed 4096 Dec  6 19:51 google

Task1/META_INF/com/google:
total 4
drwxrwxr-x 2 seed seed 4096 Dec  6 19:57 android

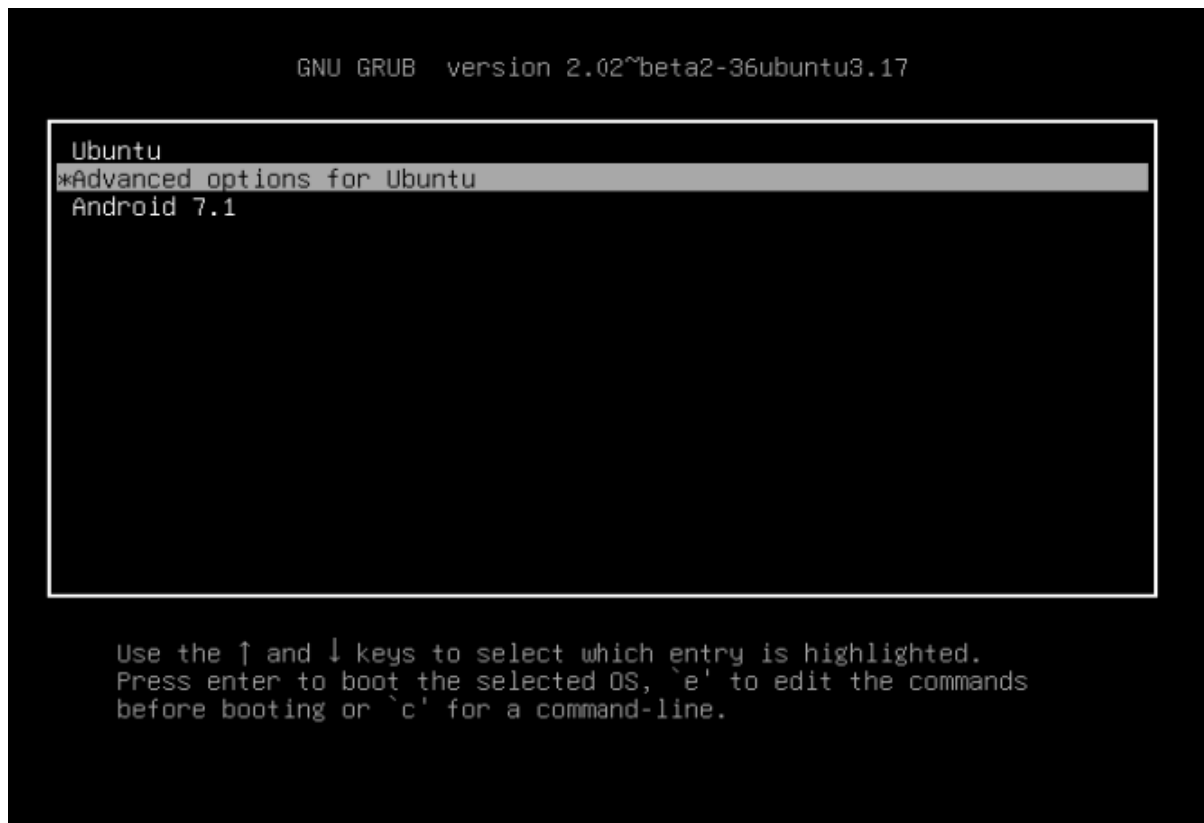
Task1/META_INF/com/google/android:
total 8
-rw-rw-r-- 1 seed seed  27 Dec  6 19:48 dummy.sh
-rwxrwxr-x 1 seed seed 144 Dec  6 19:56 update-binary
[12/06/2019]Chirag@VM:~/.../Lab12-Rooting$
```

The directory structure with the relevant files with relevant permissions.

```
[12/06/2019]Chirag@VM:~/.../Lab12-Rooting$ 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 43%)
[12/06/2019]Chirag@VM:~/.../Lab12-Rooting$
```

We zip the entire directory tree into a zip file.

Next we boot the android device into the recovery OS.



Next we connect to the Recovery OS.

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

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage
seed@recovery:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:7f:a5:46
          inet addr:10.0.2.78  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe7f:a546/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:283 errors:0 dropped:0 overruns:0 frame:0
          TX packets:29 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:24700 (24.7 KB)  TX bytes:2938 (2.9 KB)

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

We use scp to securely copy the zipfile to the /tmp folder in the android machine.

```
[12/06/2019]Chirag@VM:~/.../Lab12-Rooting$ ls
Task1 task1.zip
[12/06/2019]Chirag@VM:~/.../Lab12-Rooting$ 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+nmbyA0avocrLHpQPiGRIZknAWmJli5y06vrsA.
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
[12/06/2019]Chirag@VM:~/.../Lab12-Rooting$
```

```
seed@recovery:~$ ll /tmp/task1.zip
-rw-rw-r-- 1 seed seed 1405 Dec  6 21:25 /tmp/task1.zip
seed@recovery:~$
```

We unzip the package and run the update binary script.

One rebooting the system, Here we see that the dummy file has been created.

```
Window 1
x86_64:/ $ ls -l system/test
-rw----- 1 root root 6 2019-12-07 02:41 system/test
x86_64:/ $
```

When the init file is the first one to be invoked when the system starts and runs with root privilege, hence our task gets executed and the file is created.

Task 2: Including code via app_process

Code:

```
#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;
}
```

Application.mk

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

Android.mk

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := <compiled binary name>
LOCAL_SRC_FILES := <all source files>
include $(BUILD_EXECUTABLE)
```

We compile our code using the native compiler as shown below.

```

[12/06/2019]Chirag@VM:~/.../Task2$ ndk-build NDK_APPLICATION_MK=./Application.mk
Install      : code => libs/x86/code
[12/06/2019]Chirag@VM:~/.../Task2$ lr
.:
Android.mk  Application.mk  code.c  libs  obj

./libs:
x86

./libs/x86:
code

./obj:
local

./obj/local:
x86

./obj/local/x86:
code  objs

./obj/local/x86/objs:
code

./obj/local/x86/objs/code:
code.o  code.o.d
[12/06/2019]Chirag@VM:~/.../Task2$
```

Write the update script and build the OTA package

```

[12/06/2019]Chirag@VM:~/.../android$ ll
total 12
-rwxr-xr-x 1 seed seed 5116 Dec  6 22:59 code
-rwxr-xr-x 1 seed seed  166 Dec  6 23:02 update-binary
[12/06/2019]Chirag@VM:~/.../android$ cat update-binary
mv /android/system/bin/app_process64 /android/system/bin/app_process64_original
cp code /android/system/bin/app_process64
chmod a+x /android/system/bin/app_process64
[12/06/2019]Chirag@VM:~/.../android$
```

We write an OTA package and update-binary script for the same.

We create the appropriate directory tree and zip the package and send it to the android machine.

```

[12/06/2019]Chirag@VM:~/.../Task2$ 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/code (deflated 72%)
adding: Task2/META-INF/com/google/android/update-binary (deflated 57%)
```

```
[12/06/2019]Chirag@VM:~/.../Task2$ scp task2.zip seed@10.0.2.78:/tmp
seed@10.0.2.78's password:
task2.zip
[12/06/2019]Chirag@VM:~/.../Task2$
```

On the android machine, we unzip the file and run the update binary script as shown below.

```
android [Running] - Oracle VM VirtualBox
seed@recovery:/tmp/Task2/META_INF/com/google/android$ sudo ./update-binary
seed@recovery:/tmp/Task2/META_INF/com/google/android$ ll /android/system/bin/app_process64*
-rwxr-xr-x 1 root root 5116 Dec  6 23:16 /android/system/bin/app_process64*
-rwxr-xr-x 1 root root 5116 Dec  6 23:15 /android/system/bin/app_process64_original*
seed@recovery:/tmp/Task2/META_INF/com/google/android$ _
```

We run the update-binary script and reboot the device.

Now we see that a file dummy2 has been created in the system folder of the android device on booting. We cannot read the device since the permission is denied, however we have created a new restricted file via app process.

```
x86_64:/ $ ls -l /system/dummy2
-rw----- 1 root root 0 2019-12-09 18:45 /system/dummy2
x86_64:/ $ cat /system/dummy2
/system/bin/sh: cat: /system/dummy2: Permission denied
1|x86_64:/ $
```

When the device boots, the program app_process after init using root privilege. The app process starts the zygote daemon which has the task of starting applications. Every app that starts forks from the zygote. We modify the app process to launch a malicious app along with the zygote. Hence our script creates a file dummy2 along with the rest of the apps and processes.

Task 3

Implement SimpleSu for getting root shell

We download and unzip the file from the lab website.

```

[12/09/2019]Chirag@VM:~/.../Task3$ ls
SimpleSU.zip
[12/09/2019]Chirag@VM:~/.../Task3$ 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
[12/09/2019]Chirag@VM:~/.../Task3$
```

We build the program to obtain root shell using the compile_all shell script

```

[12/09/2019]Chirag@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////////

[12/09/2019]Chirag@VM:~/.../SimpleSU$ ls -lR ./*/x86
./mydaemon/libs/x86:
total 12
-rwxr-xr-x 1 seed seed 9232 Dec  9 20:52 mydaemon

./mysu/libs/x86:
total 12
-rwxr-xr-x 1 seed seed 9232 Dec  9 20:52 mysu
[12/09/2019]Chirag@VM:~/.../SimpleSU$ l
```

We then copy these files into the android folder of the relevant working directory tree.

We have to modify the update binary file such that it runs mydaemon as the app_process64 and have to place the mysu executable in the default directory for executables, i.e. /android/system/xbin directory.

```
/bin/bash 108x46
1 mv /android/system/bin/app_process64 /android/system/bin/app_process_original
2 cp mydaemon /android/system/bin/app_process64
3 chmod a+x /android/system/bin/app_process64
4 cp mysu /android/system/xbin/
5 chmod a+x /android/system/xbin/mysu
```

```
[12/09/2019]Chirag@VM:~/../SimpleSU$ ls -lR Task3/
Task3/:
total 4
drwxrwxr-x 3 seed seed 4096 Dec  9 21:08 META_INF

Task3/META_INF:
total 4
drwxrwxr-x 3 seed seed 4096 Dec  9 21:08 com

Task3/META_INF/com:
total 4
drwxrwxr-x 3 seed seed 4096 Dec  9 21:08 google

Task3/META_INF/com/google:
total 4
drwxrwxr-x 2 seed seed 4096 Dec  9 21:59 android

Task3/META_INF/com/google/android:
total 28
-rwxr-xr-x 1 seed seed 9232 Dec  9 21:09 mydaemon
-rwxr-xr-x 1 seed seed 9232 Dec  9 21:09 mysu
-rwxrwxr-x 1 seed seed  234 Dec  9 21:59 update-binary
[12/09/2019]Chirag@VM:~/../SimpleSU$
```

We zip the file and send it to the recovery os.

```
/bin/bash 108x46
[12/09/2019]Chirag@VM:~/../SimpleSU$ zip -r task3.zip Task3/
  adding: Task3/ (stored 0%)
  adding: Task3/META_INF/ (stored 0%)
  adding: Task3/META_INF/com/ (stored 0%)
  adding: Task3/META_INF/com/google/ (stored 0%)
  adding: Task3/META_INF/com/google/android/ (stored 0%)
  adding: Task3/META_INF/com/google/android/update-binary (deflated 61%)
  adding: Task3/META_INF/com/google/android/mydaemon (deflated 60%)
  adding: Task3/META_INF/com/google/android/mysu (deflated 60%)
[12/09/2019]Chirag@VM:~/../SimpleSU$ scp task3.zip seed@10.0.2.78:/tmp
seed@10.0.2.78's password:
task3.zip                                                                100% 9001
[12/09/2019]Chirag@VM:~/../SimpleSU$
```

We extract the files and run the update binary script

```
android (Snapshot 1) [Running] - Oracle VM VirtualBox
seed@recovery:~$ cd /tmp/
seed@recovery:/tmp$ unzip task3.zip
Archive:  task3.zip
  creating: Task3/
  creating: Task3/META_INF/
  creating: Task3/META_INF/com/
  creating: Task3/META_INF/com/google/
  creating: Task3/META_INF/com/google/android/
  inflating: Task3/META_INF/com/google/android/update-binary
  inflating: Task3/META_INF/com/google/android/mydaemon
  inflating: Task3/META_INF/com/google/android/mysu
seed@recovery:/tmp$ ls -lR Task3/
Task3/:
total 4
drwxrwxr-x 3 seed seed 4096 Dec  9 21:08 META_INF

Task3/META_INF:
total 4
drwxrwxr-x 3 seed seed 4096 Dec  9 21:08 com

Task3/META_INF/com:
total 4
drwxrwxr-x 3 seed seed 4096 Dec  9 21:08 google

Task3/META_INF/com/google:
total 4
drwxrwxr-x 2 seed seed 4096 Dec  9 21:59 android

Task3/META_INF/com/google/android:
total 28
-rwxr-xr-x 1 seed seed 9232 Dec  9 21:09 mydaemon
-rwxr-xr-x 1 seed seed 9232 Dec  9 21:09 mysu
-rwxrwxr-x 1 seed seed  234 Dec  9 21:59 update-binary
seed@recovery:/tmp$ _
```

```
android (Snapshot 1) [Running] - Oracle VM VirtualBox
seed@recovery:/tmp/Task3/META_INF/com/google/android$ sudo ./update-binary
seed@recovery:/tmp/Task3/META_INF/com/google/android$ ll /android/system/bin/app_process*
lrwxrwxrwx 1 root root    13 Mar 29 2018 /android/system/bin/app_process -> app_process64*
-rwxr-xr-x 1 root root 2000 17948 Mar 29 2018 /android/system/bin/app_process32*
-rwxr-xr-x 1 root root 9232 Dec  9 22:11 /android/system/bin/app_process64*
-rwxr-xr-x 1 root root 2000 22720 Mar 29 2018 /android/system/bin/app_process_original*
seed@recovery:/tmp/Task3/META_INF/com/google/android$ ll /android/system/sbin/m
man                micro_bench_static64  mkswap             mountpoint
matchpathcon       mkdir                mktemp             mpstat
md5sum             mke2fs              mmc_utils          mv
mesg               mkfifo              modinfo            mysu
micro_bench        mkfs.ext2            modprobe
micro_bench64      mkfs.vfat            more
micro_bench_static mknod                mount
seed@recovery:/tmp/Task3/META_INF/com/google/android$ ll /android/system/sbin/mysu
-rwxr-xr-x 1 root root 9232 Dec  9 22:11 /android/system/sbin/mysu*
seed@recovery:/tmp/Task3/META_INF/com/google/android$ _
```

We reboot the system and launch the terminal

Here we start the app mysu, which generates a root shell as shown below.

```
Window 1
x86_64:/ $ whoami
u0_a36
x86_64:/ $ mysu
WARNING: linker: /system/xbin/mysu has text relocations. This is wasting memory and prevents 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:/ # whoami
root
x86_64:/ #
```

```
android (Snapshot 1) [Running] - Oracle VM VirtualBox
Window 1
endar
u0_a25    2352  1040  1170728 137616    0 0000000000 S com.android.vending
u0_a54    2465  1040  1137612 109064    0 0000000000 S com.google.android.gm
u0_a56    2560  1040  1077596 77216    0 0000000000 S com.google.process.gapps
u0_a17    2660  1040  1168596 136424    0 0000000000 S com.google.android.gms.u
nstable
u0_a17    2746  1040  1137092 111416    0 0000000000 S com.google.android.gms.u
i
u0_a25    2868  1040  1107732 97524    0 0000000000 S com.android.vending:downl
oad_service
u0_a36    2924  1041  1104940 129996    0 0000000000 S jackpal.androidterm
u0_a67    2944  1040  1072120 71080    0 0000000000 S com.android.printspooler
u0_a36    2964  2924  8316    2712    0 0000000000 S /system/bin/sh
u0_a36    2980  2964  5064    1900    0 0000000000 S mysu
root      2981  1069  8316    2908    0 0000000000 S /system/bin/sh
root      3036  2981  9880    2720    0 0000000000 R ps
x86_64:/ # ps | grep mysu
u0_a36    2980  2964  5064    1900    0 0000000000 S mysu
x86_64:/ # ls /proc/2980/fd
0 1 2 3
x86_64:/ # ls /proc/2964/fd
0 1 10 2
x86_64:/ # ls /proc/2944/fd
0 10 12 14 16 18 2 21 23 25 27 29 4 6 8
1 11 13 15 17 19 20 22 24 26 28 3 5 7 9
x86_64:/ #
```

Here our malicious mydaemon starts with root privileges hence gives us control, we exploit this property.

Here we see that the file descriptors for mysu are different than /system/bin/sh.

Thus a root shell has been obtained from android.

Q&A

- Server launches the original app process binary

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

- Client sends its FDs

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

- Server forks to a child process

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

- Child process receives client's FDs

```
int child_process(int socket, char** argv){  
    //handshake  
    handshake_server(socket);  
  
    int client_in = recv_fd(socket);  
    int client_out = recv_fd(socket);  
    int client_err = recv_fd(socket);
```

- Child process redirects its standard I/O FDs

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

- Child process launches a root shell

```
//launch default shell directly  
char* shell[] = {"/system/bin/sh", NULL};  
execve(shell[0], shell, NULL);  
return (EXIT_SUCCESS);
```

Appendix:

Codes:

Mysu.c:

```
/*
 * File:   mysu.c
 * Author: Zhuo Zhang, Syracuse University
 *        zzhan38@syr.edu
 *
 * Version: 1.0
 * Release Date: 1/30/2016
 */

/* Version 1.0 - First Release
 * This project is a client
 * It ask the daemon server to launch a root shell for it
 * It will pass its STDIN, STDOUT, STDERR file descriptors to the server via U
nix domain socket
 */

/* This project is based on open source su project
 * Source: https://github.com/koush/Superuser
 * Original License:
 * ** Copyright 2010, Adam Shanks (@ChainsDD)
 * ** Copyright 2008, Zinx Verituse (@zinxv)
 * **
 * ** Licensed under the Apache License, Version 2.0 (the "License");
 * ** you may not use this file except in compliance with the License.
 * ** You may obtain a copy of the License at
 * **
 * **      http://www.apache.org/licenses/LICENSE-2.0
 * **
 * ** Unless required by applicable law or agreed to in writing, software
 * ** distributed under the License is distributed on an "AS IS" BASIS,
 * ** WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implie
d.
 * ** See the License for the specific language governing permissions and
 * ** limitations under the License.
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>      //socket() bind() listen() accept() AF_UNIX
#include <fcntl.h>           //fcntl()
#include <string.h>          //strerror()
#include <errno.h>           //errno
```

```

#include <sys/un.h>                //struct sockaddr_un

#include "../socket_util/socket_util.h"
#include "../server_loc.h"

#define ERRMSG(msg) fprintf(stderr, "%s", msg)

#define DEFAULT_SHELL "/system/bin/sh"

#define SHELL_ENV "SHELL=" DEFAULT_SHELL
#define PATH_ENV "PATH=/system/bin:/system/sbin"

//try to connect to the server and get a socket file descriptor
int config_socket() {

    struct sockaddr_un sun;

    //create socket fd
    int socket_fd = socket(AF_UNIX, SOCK_STREAM, 0);
    if (socket_fd < 0) {
        ERRMSG("failed to create socket fd\n");
        exit (EXIT_FAILURE);
    }

    //set the socket file descriptor
    //with flag FD_CLOEXEC, socket_fd will stay valid through fork()
    //but will be destroyed by all exec family functions (e.g. execve())
    if (fcntl(socket_fd, F_SETFD, FD_CLOEXEC)) {
        ERRMSG("failed on fcntl\n");
        exit (EXIT_FAILURE);
    }

    //set struct sockaddr_un
    /*
        struct sockaddr_un {
            sa_family_t sun_family;           //AF_UNIX
            char        sun_path[108];        //pathname
        };
    */
    memset(&sun, 0, sizeof(sun));
    sun.sun_family = AF_UNIX;
    strncpy(sun.sun_path, SERVER_LOC, sizeof(sun.sun_path));

    //connect to server
    if (0 != connect(socket_fd, (struct sockaddr*)&sun, sizeof(sun))) {
        ERRMSG("failed to connect server\n");
        exit (EXIT_FAILURE);
    }
}

```



```

    return socket_fd;
}

//try to connect the daemon server
//pass stdin, stdout, stderr to server
//hold the session to operate the root shell created and linked by server
int connect_daemon() {

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

    //do handshake
    handshake_client(socket);

    ERRMSG("sending file descriptor \n");
    fprintf(stderr, "STDIN %d\n", STDIN_FILENO);
    fprintf(stderr, "STDOUT %d\n", STDOUT_FILENO);
    fprintf(stderr, "STDERR %d\n", STDERR_FILENO);

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

    //hold the session until server close the socket or some error occurs
    //in my design, server should not send things back through socket after ha
ndshake
    //read() function will block the process, thus we hold the session
    //if the socket is closed, read() will return 0
    //or error occurs, read() will return a negative integer
    char dummy[2];
    ERRMSG("2 \n");
    int flag = 0;
    do {
        flag = read(socket, &dummy, 1);
    } while (flag > 0);

    ERRMSG("3 \n");

    close(socket);

    //print out error message if has
    if (flag < 0) {
        ERRMSG("Socket failed on client: ");
        ERRMSG(strerror(errno));
        ERRMSG("\n");
        return (EXIT_FAILURE);
    }
}

```



```
    return (EXIT_SUCCESS);
}

int main(int argc, char** argv) {
    //if not root
    //connect to root daemon for root shell
    if (getuid() != 0 && getgid() != 0) {
        ERRMSG("start to connect to daemon \n");

        return connect_daemon();
    }
    //if root
    //launch default shell directly
    char* shell[] = {"/system/bin/sh", NULL};
    execve(shell[0], shell, NULL);
    return (EXIT_SUCCESS);
}
```

Mydaemon.c

```
/*
 * File:   mydaemonsu.c
 * Author: Zhuo Zhang, Syracuse University
 *         zzhan38@syr.edu
 *
 * Version: 2.1
 * Release Date: 3/11/2016
 */

/* Version 2.1
 * Instead of copying argv into a new buffer
 * this version just modify argv[0] and use the argv to pass all the arguments
 * Previously I thought argv is not NULL-terminated, but in fact it is
 */

/* Version 2.0
 * From this version, I use an approach to hack app_process
 * Thus we don't need to rely on outer script file like init.sh to launch this
 daemon
 * All we need to do is to link app_process to this binary file
 * and rename the original app_process32 or app_process64 to app_process_origi
nal
 */

/* Version 1.0 - First Release
 * This project is a server
 * It runs under root privilege and wait for client's connect
 * After connection, it launch a terminal for the client and redirect
 * the terminal's input and output to the client
 */

/* This project is based on open source su project
 * Source: https://github.com/koush/Superuser
 * Original License:
 * ** Copyright 2010, Adam Shanks (@ChainsDD)
 * ** Copyright 2008, Zinx Verituse (@zinxv)
 * **
 * ** Licensed under the Apache License, Version 2.0 (the "License");
 * ** you may not use this file except in compliance with the License.
 * ** You may obtain a copy of the License at
 * **
 * ** http://www.apache.org/licenses/LICENSE-2.0
 * **
 * ** Unless required by applicable law or agreed to in writing, software
 * ** distributed under the License is distributed on an "AS IS" BASIS,
```

```

*    ** WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
*    ** See the License for the specific language governing permissions and
*    ** limitations under the License.
*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>    //socket() bind() listen() accept() AF_UNIX
#include <fcntl.h>         //fcntl()
#include <string.h>        //strerror()
#include <errno.h>         //errno
#include <sys/un.h>        //struct sockaddr_un
#include <sys/stat.h>      //umask() mkdir()
#include <stdbool.h>       //bool true false

#include "../socket_util/socket_util.h"
#include "../server_loc.h"

#define ERRMSG(msg) fprintf(stderr, "%s", msg)

#define DEFAULT_SHELL "/system/bin/sh"

#define SHELL_ENV "SHELL=/system/bin/sh"
#define PATH_ENV "PATH=/system/bin:/system/sbin"

#define APP_PROCESS "/system/bin/app_process_original"

extern char** environ;

//create a UNIX domain socket and return its file descriptor
int creat_socket() {
    int socket_fd;
    struct sockaddr_un sun;

    //open socket
    socket_fd = socket(AF_UNIX, SOCK_STREAM, 0);
    if (socket_fd < 0) {
        ERRMSG("failed to open socket\n");
        exit(EXIT_FAILURE);
    }

    //set the socket file descriptor
    //with flag FD_CLOEXEC, socket_fd will stay valid through fork()
    //but will be destroyed by all exec family functions (e.g. execve())
    if (fcntl(socket_fd, F_SETFD, FD_CLOEXEC)) {

```

```

        ERRMSG("failed to fcntl\n");
        goto err;
    }

    //set struct sockaddr_un
    /*
        struct sockaddr_un {
            sa_family_t sun_family;           //AF_UNIX
            char        sun_path[108];        //pathname
        };
    */
    memset(&sun, 0, sizeof(sun));
    sun.sun_family = AF_UNIX;
    strncpy(sun.sun_path, SERVER_LOC, sizeof(sun.sun_path));

    //get rid of potential existing file due to previous error
    unlink(sun.sun_path);
    unlink(SERVER_DIR);

    //backup current umask
    //and change umask to allow all permissions
    int previous_umask = umask(0);

    //make new server path
    mkdir(SERVER_DIR, 0777);

    //bind socket
    if (bind(socket_fd, (struct sockaddr*)&sun, sizeof(sun)) < 0) {
        ERRMSG("failed to bind socket\n");
        goto err;
    }

    //restore umask
    umask(previous_umask);

    //start listening on the socket
    if (listen(socket_fd, 10) < 0) {
        ERRMSG("failed to listen\n");
        goto err;
    }

    return socket_fd;

err:
    close(socket_fd);
    exit(EXIT_FAILURE);
}

```

```

//the code executed by the child process
//it launches default shell and link file descriptors passed from client side
int child_process(int socket, char** argv){
    //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
    chdir("/");

    char* env[] = {SHELL_ENV, PATH_ENV, NULL};
    char* shell[] = {DEFAULT_SHELL, NULL};

    execve(shell[0], shell, env);

    //expect no return from execve
    //only if execve fails
    ERRMSG("Failed on launching shell: ");
    ERRMSG(strerror(errno));
    ERRMSG("\n");

    close(socket);

    exit(EXIT_FAILURE);
}

//start the daemon and keep waiting for connections from client
void run_daemon( char** argv) {
    if (getuid() != 0) {
        ERRMSG("Daemon require root privilege\n");
        exit(EXIT_FAILURE);
    }

    //get a UNIX domain socket file descriptor
    int socket = creat_socket();

    //wait for connection
    //and handle connections
    int client;
    while ((client = accept(socket, NULL, NULL)) > 0) {

```

```

        if (0 == fork()) {
            close(socket);
            ERRMSG("Child process start handling the connection\n");
            exit(child_process(client,argv));
            child_process(client, argv);
        }
        else {
            close(client);
        }
    }

    //expect daemon never end execution
    //unless socket failed
    ERRMSG("Daemon quits: ");
    ERRMSG(strerror(errno));
    ERRMSG("\n");

    close(socket);
    close(client);

    exit(EXIT_FAILURE);
}

//try to connect to the daemon to determine whether it is running
bool detect_daemon() {

    struct sockaddr_un sun;

    //create socket fd
    int socket_fd = socket(AF_UNIX, SOCK_STREAM, 0);
    if (socket_fd < 0) {
        ERRMSG("failed to create socket fd\n");
        exit (EXIT_FAILURE);
    }

    //set socket fd
    if (fcntl(socket_fd, F_SETFD, FD_CLOEXEC)) {
        ERRMSG("failed on fcntl\n");
        exit (EXIT_FAILURE);
    }

    //set sun
    memset(&sun, 0, sizeof(sun));
    sun.sun_family = AF_UNIX;
    strncpy(sun.sun_path, SERVER_LOC, sizeof(sun.sun_path));

    //connect to server
    //return false if connection failed (daemon is not running)

```

```
    if (0 != connect(socket_fd, (struct sockaddr*)&sun, sizeof(sun))) {  
        return false;  
    }  
  
    //close the socket and return true if connection succeeded (daemon is running)  
    close(socket_fd);  
    return true;  
}  
  
int main(int argc, char** argv) {  
    pid_t pid = fork();  
    if (pid == 0) {  
        //initialize the daemon if not running  
        if (!detect_daemon())  
            run_daemon(argv);  
    }  
    else {  
        argv[0] = APP_PROCESS;  
        execve(argv[0], argv, environ);  
    }  
}
```