Post-Lab 4 Report
Secure Systems and Networks
dr inż. Tomasz Surmacz

Piotr Giedziun 184731
Lab Date: 13/01/15

Abstract

This report describes my attempts to perform reverse engineering on executable file. Task was accomplished on both debug and release application. Following applications were used: gdb, objdump and hopper disassembler v3. Paper covers brute-force and reverse engineering approaches.

Introduction

This time around, instead of writing a code, we were ask to perform reverse engineering on existing one. For given ELF 64-bit LSB executable we had to find out the code. Application had some kind of GUI where user were asked to enter his student id and password. Access was granted only when username and password was valid. We were also told that application have internal password generator. In addition to that, we also received one pair of valid data (0 - student id, 12345678 - password).

Reverse engineering

I will try to explain shortly explain myself before I will share my first thoughts on this task. My assumption was that password is hashed, so I used the same mindset as for MD5 hash cracking. I decided that I will start by trying to brute-force access. It's the simplest way to achieve our goal. At least I thought so. Due to the fact that my further assumptions was invalid I decided to bury this solution.
Brute-force approach failed due to following reasons:

- I assumed that password always consists of 8 digital characters.
- There is $10^8 - 10^7$ possible solutions.
- Execution of one test took about 1 seconds.
- Total time of execution estimated to roughly 2 years (random probability)

Using expect and python I was quickly able to prepare application.

```
#!/usr/bin/expect
spawn ./keygen-en

expect "USER: "
send "[lindex $argv 0]\r"
expect "KEY: "
send "[lindex $argv 1]\r"
interact
```

Listing 1: Execute keygen-en application and inject inputs

```python
#!/usr/bin/python
from subprocess import Popen, PIPE

def execute(username, passwd):
        process = Popen(["./single_test.exp", str(username), str(passwd)], stdout=PIPE)
        (output, err) = process.communicate()
        exit_code = process.wait()
        return "  _____             __    _      __   __" in output

for passwd in xrange(10000000, 99999999):
        if passwd % 1000 == 0:
                print "checking", passwd
        if execute(184731, passwd):
                print "pass", passwd
                return
```

Listing 2: Full search

Knowing that "Act first, think later" approach failed, I decided to decompile application. Using objdump I was able to receive application source code.

```
$ wget http://dream.ict.pwr.wroc.pl/ssn/keygen/keygen-en-stripped
$ objdump -S keygen-en-symbols > code.s
```

I knew that at some point generated password will be compared with user input. At some point generated password will be stored in one of available registers. I did some research about gdb instructions, there is a way to set a watcher for variable/register with expression. Each time expression will be true, application will stop.

My idea was to watch all registers, at some point application should break. Next step would be to display all registers values (i r) and find out where it's stored. I can do such thing, because I know password for student ID equal to 0. Anyway, It's not working. Somehow it's causing SEGFAULT (it's working fine for integers).

```
b __libc_start_main
r
watch $rbp if strcmp($rbp, "12345678") == 0
watch $rsi if strcmp($rsi, "12345678") == 0
...
c
```

In obtained source code (.text section) I was looking for calls to functions like fgets, getchar, scanf etc. After gathering input data application have to invoke password generator, my idea was to find out memory address. Next step would be to find generator function virtual address in disassembled source file. I was instructed to skip password generation function, instead search for logic that will compare result with user input. From this point it's really easy to perform.

```
4008ce:       48 89 ce                mov    %rcx,%rsi
4008d1:       48 89 c7                mov    %rax,%rdi
4008d4:       e8 97 fd ff ff          callq  400670 <strncmp@plt>
4008d9:       89 45 f8                mov    %eax,-0x8(%rbp)
4008dc:       83 7d f8 00             cmpl   $0x0,-0x8(%rbp)
4008e0:       75 0c                   jne    4008ee <fflush@plt+0x20e>
```

I have found strcmp usage (function compares two strings). Linux is using virtual memory, so address 0x4008d4 will be always the same. Now, all I had to do was to debbug application and check arguments of strcmp function.

```
gdb keygen/keygen-en-stripped
(gdb) b *0x4008d4
(gdb) r
(gdb) printf "%s\n", $rdi
a
(gdb) printf "%s\n", $rsi
7vrf4kin
```

My student ID is 184731, generated password - **7vrf4kin**.

Gen function

Using hooper I was able to decompile gen function source.

```
int gen(int arg0, int arg1, int arg2) {
    int var_24 = LODWORD(arg0);
    int var_30 = arg1;
    int var_38 = arg2;
    char[] var_10 = "lrjz1ust0obix7vp6ym9kcd4q3egw8an5h2fxyz";
    char[] var_18 = "2swx9loh3dec8nik4frv7jum5gtb1aqz06py";
    if ((var_24 == 0x0) || (var_24 == 0x1)) {
            var_18 = *tab0;
    }
    var_4 = 0x0;
    do {
            LODWORD(rax) = var_4;
            if (rax >= var_38) {
                break;
            }
            ...
            temp_2 = LOWORD(var_3C) * LODWORD(0x51eb851f);
            if (LODWORD(LODWORD(LODWORD(HIDWORD(temp_2)) >> 0x5) & 0x1) == 0x0) {
                    *(int8_t *)(var_4 + var_30) = LOBYTE(
                        *(int8_t *)(var_8 + var_18) & 0xff);
            }
            else {
                    *(int8_t *)(var_4 + var_30) = LOBYTE(*(int8_t *)(
                        LODWORD(LODWORD(0x23) - var_8) + var_18) & 0xff);
            }
            var_4 = var_4 + 0x1;
    } while (true);
    return rax;
}
```

Listing 3: Password gen function

In my case password is generated from var_18, reverse order of my password is "nik4fr". It's part of var_18. It's seems like code is reading these buffers in both orders (depending on student ID), starting from point that's also calculated from student ID.

References

[1] http://ftp.gnu.org/old-gnu/Manuals/gdb/html_node/gdb_28.html

[2] http://www.hopperapp.com/

[3] http://dream.ict.pwr.wroc.pl/ssn/index.en.html