HW-0x08 Writeup

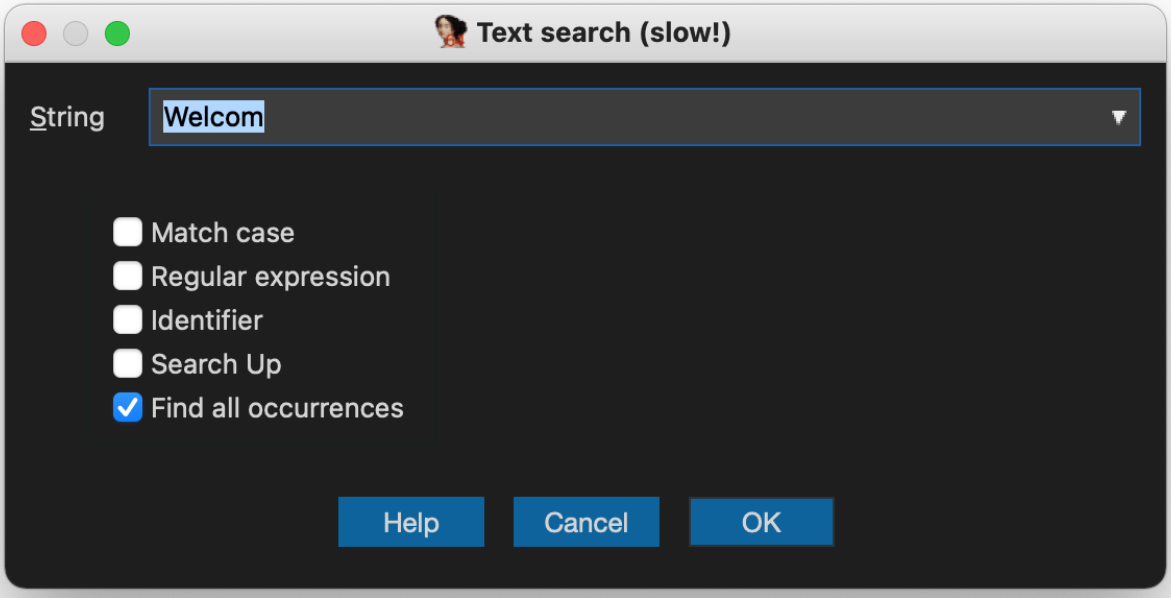
School/Grade: 交大資科工所 碩一

Student ID: 309551004 (王冠中)

ID: aesophor

wishMachine (250 pts)

我們可以搜尋 "Welcom" 字串來找到它在哪裡被使用到，而經過搜尋可以發現是 `sub_400B8D()` 。



Address	Function	Instruction
sub_400B8D:5 (400B8D)	sub_410BB0("Welcom to t…	

```
__int64 sub_400B8D()
{
  __int64 result; // rax

  sub_410BB0("Welcom to the wish machine");
  sub_410BB0("Give me 1000 coins, and I'll make your wish come true");
  sub_410BB0("Each coin has a serial number on it, show me the number");
  result = sub_44A300(0);
  if ( result == -1 )
    sub_40F130(0LL, 0LL);
  return result;
}
```

但這個程式會呼叫 ptrace 這個 syscall，導致我們在正常的情況下無法用 gdb 跟進去 debug。有兩個方法，一個是 patch binary，另外一個方法是讓 呼叫 ptrace 的那個 syscall 指令執行完之後，就用 gdb 把 rax 改成 0，這樣就可以 bypass ptrace

接下來同樣用字串搜尋，去搜尋 "One by one" 字串，可以找到 `sub_400BE6()` 。這個 loop 會跑 1000 次，每次都問你第 i 個金幣的 serial number，只要讓 loop 順利跑過就會得到 flag。

```
  for ( i = 0; i <= 999; ++i )
  {
    printf((unsigned __int64)"One by one, What is the serial number of coin%d ?");
    memset(&v2, 0LL, 70LL);
    scanf((unsigned __int64)"%70s");
    func1((__int64)&v2);
    func2(&v3, &v2);
  }
  printf((unsigned __int64)"Ok, the flag is %s");
  result = 0LL;
  if ( __readfsqword(0x28u) != v74 )
    sub_44BD80();
  return result;
}
```

func1：做一連串的運算，最後又 call 一個 function pointer

```
__int64 __fastcall func1(__int64 a1)
{
  __int64 result; // rax
  signed int i; // [rsp+1Ch] [rbp-4h]

  for ( i = 0; i <= 69; i += step )
  {
    store_arg1_addr = a1;
    str_offset = *((_DWORD *)&unk_6D5114 + 10 * base);
    step = *((_DWORD *)&unk_6D5118 + 10 * base);
    s1 = *((_DWORD *)&unk_6D511C + 10 * base);
    s2 = secret_arr[10 * base];
    func_offset = *((_DWORD *)&unk_6D5110 + 10 * base);
    call_func = *((_QWORD *)&unk_6D5100 + 5 * base) + func_offset;
    ((void (*)(void))call_func)();
    ++base;
    result = (unsigned int)step;
  }
  return result;
}
```

func2：跟 `main()` 裡的東西進行 XOR

```
_BYTE *__fastcall func2(__int64 a1, __int64 a2)
{
  _BYTE *result; // rax
  signed int i; // [rsp+1Ch] [rbp-4h]

  for ( i = 0; i <= 69; ++i )
  {
    result = (_BYTE *)(i + a1);
    *result ^= *(_BYTE *)(i + a2);
  }
  return result;
}
```

接下來實際跑跑看，看看 `call_func` 這個 function pointer 會是什麼，實際跑起來會發現只有 5 種結果。這邊的 function pointer 其實就是在檢查你提供的 serial number 是否正確。

5 種可能：0x4011d6、0x40102d、0x401138、0x4010c8、0x400fbe，這邊我分別把他們重命名為 `check1()` ~ `check5()`

```
__int64 check1()
{
  __int64 result; // rax
  int v1; // [rsp+Ch] [rbp-14h]
  int i; // [rsp+10h] [rbp-10h]
  int v3; // [rsp+14h] [rbp-Ch]
  signed int v4; // [rsp+18h] [rbp-8h]
  signed int j; // [rsp+1Ch] [rbp-4h]

  for ( i = 0; ; ++i )
  {
    result = (unsigned int)step;
    if ( i >= step )
      break;
    v3 = 0;
    v4 = 1;
    for ( j = 0; j < *(char *)(store_arg1_addr + str_offset + i); ++j )
    {
      v1 = v3 + v4;
      v3 = v4;
      v4 = v1;
    }
    if ( v1 != *((_DWORD *)&store_arg1_addr + i + 4LL + 1) )
      sub_40F130(0LL);
  }
  return result;
}
```

```
__int64 check2()
{
  __int64 result; // rax
  int v1; // [rsp+4h] [rbp-Ch]
  int i; // [rsp+8h] [rbp-8h]
  signed int j; // [rsp+Ch] [rbp-4h]

  for ( i = 0; ; ++i )
  {
    result = (unsigned int)step;
    if ( i >= step )
      break;
    v1 = 0;
    for ( j = 0; j < *(char *)(store_arg1_addr + str_offset + i); ++j )
    {
      if ( j & 1 )
        v1 += 2;
      else
        v1 += 11;
    }
    if ( v1 != *((_DWORD *)&store_arg1_addr + i + 4LL + 1) )
      sub_40F130(0LL);
  }
  return result;
}
```

```
__int64 check3()
{
  __int64 result; // rax
  signed int v1; // [rsp+4h] [rbp-Ch]
  int i; // [rsp+8h] [rbp-8h]
  signed int j; // [rsp+Ch] [rbp-4h]

  for ( i = 0; ; ++i )
  {
    result = (unsigned int)step;
    if ( i >= step )
      break;
    v1 = -88035316;
    for ( j = 0; j < *(char *)(store_arg1_addr + str_offset + i); ++j )
    {
      if ( j & 1 )
        v1 -= 120;
      else
        v1 -= 30600;
    }
    if ( v1 != *((_DWORD *)&store_arg1_addr + i + 4LL + 1) )
      sub_40F130(0LL);
  }
  return result;
}
```

```
__int64 check4()
{
  __int64 result; // rax
  int i; // [rsp+Ch] [rbp-4h]

  for ( i = 0; ; ++i )
  {
    result = (unsigned int)step;
    if ( i >= step )
      break;
    if ( *((_DWORD *)&store_arg1_addr + i + 4LL + 1) != (*(char *)(store_arg1_addr + str_offset
      sub_40F130(0LL);
  }
  return result;
}
```

```
__int64 check5()
{
  __int64 result; // rax
  int i; // [rsp+Ch] [rbp-4h]

  for ( i = 0; ; ++i )
  {
    result = (unsigned int)step;
    if ( i >= step )
      break;
    if ( *((_DWORD *)&store_arg1_addr + i + 4LL + 1) != 135 * *(char *)(store_arg1_addr + str_o
      sub_40F130(0LL);
  }
  return result;
}
```

他們的共同點是會從 input 抓某個 byte 去比較。這邊我們可以把會用到的 memory 直接 dump 出來，然後模仿這些 function 的邏輯，就可以逆推出正確的 input 來通過檢查了。

```
$ objdump -s --start-address=0x6d5100 --stop-address=0x8a1008 ./wishMachine
```

這邊可以寫個 script 去整理剛剛 dump 出來的 memory data，把他們整理成 signed int array。接下來把上面 5 個 check function 照貼到一個 .c 檔案裡面自己進行逆推。注意在逆推 input 的時候要注意變數的 type 要和原本程式的邏輯一致，否則如果遇到 signed integer overflow 的狀況就會推錯。剩下的就是模仿程式邏輯找出 1000 次 loop 所需的正確 input 即可拿到 flag。

```
-- trimmed --
One by one, What is the serial number of coin997 ?
One by one, What is the serial number of coin998 ?
One by one, What is the serial number of coin999 ?
Ok, the flag is
FLAG{7hes3_func710n_ptrs_g1v3_m3_l0t_0o0of_f4n_I_w4n7_m00r3_11!!l1!|!}
```

flag: `FLAG{7hes3_func710n_ptrs_g1v3_m3_l0t_0o0of_f4n_I_w4n7_m00r3_11!!l1!|!}`

SecureContainProtect (250 pts)

這題要先解數讀，透過網路上的 sudoku solver 可以解出來：

```
l1l2l7l5l3l6l4l9j5h7h1h2h8hhh4h9j6ll5l4ll1ll8l3j6h9h8hh3h2h4hh1j3l6l9l8llll2l1j4h
h5h9h6hh7h8h2j5l2ll9l7l4l3llj7hh9h6h2hhh3h4j7ll6l3l1l8ll5l2z
```

底下可以看到 `byte_202E00` 、 `byte_202020` 是兩個長度都是 k 的 array。程式會 byte-wise 將 (user input) XOR (byte_202E00) XOR (byte_202020)，然後 v6 持續累加，只要讓下圖中 line 39 的 if statement 成立，使得 `v6 == 257498` 就會印出 flag。

```
27  if ( v2 == -194062 && v3 == 14763 )
28  {
29    v6 = 0;
30    printf("%3161s%159s", byte_2020E0, byte_202D40);
31    __isoc99_scanf("%39s", s);
32    for ( k = 0; k <= 6014; ++k )
33    {
34      v0 = byte_202E00[k];
35      v1 = byte_202020[k % 81];
36      byte_202E00[k] = v1 ^ s[k % strlen(s)] ^ v0;
37      v6 += byte_202E00[k];
38    }
39    if ( v6 == 257498 )
40      puts(byte_202E00);
41    else
42      puts(
43        "\n"
44        "\n"
45        "You are not the agent.....\n"
46        "Are you trying to steal our secret?\n"
47        "A well-trained killer is coming to your place for a cup of tea...");
48    exit(0);
49  }
50  printf("\n\nWrong, you are bad at sudoku.");
51  exit(0);
```

最後要輸入 Action Code。這邊我突然想到 ascii art 的特色是：用到的字不多，很多一樣的 char 會頻繁出現。

所以我讓 ascii art 跟 sudoku 的答案做 XOR，然後把結果拿去 XOR 所有的 printable character，會看到下面這個結果：

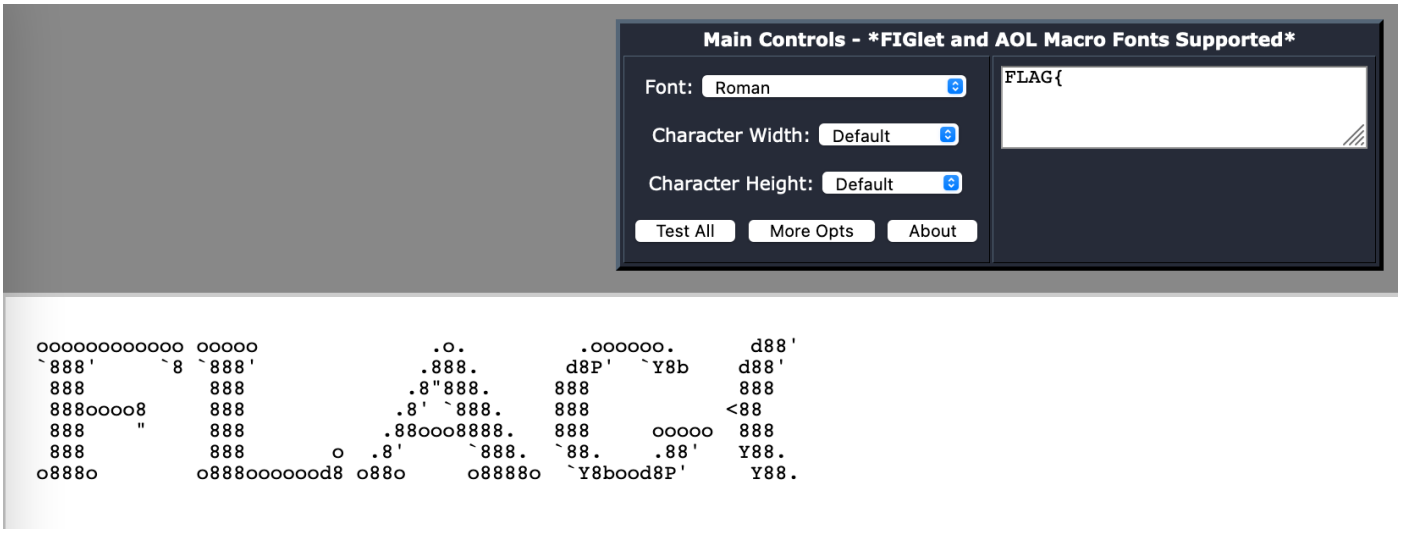
```
Key = 1e: ............Z.....PJaQXam}n.....T.SLGNJ
Key = 1f: ......./.../[.....QK`PY`l|o.....U.RMFOK
Key = 20: +*,=6?;.;'*.d ,:"*nt_of_SCP-2521j*mrypt
Key = 21: *+-<7>:.:&+.e!-;#+ou^ng^RBQ,3430k+lsxqu
```

0x20（whitespace）有： `nt_of_SCP-2521j*mrypt`

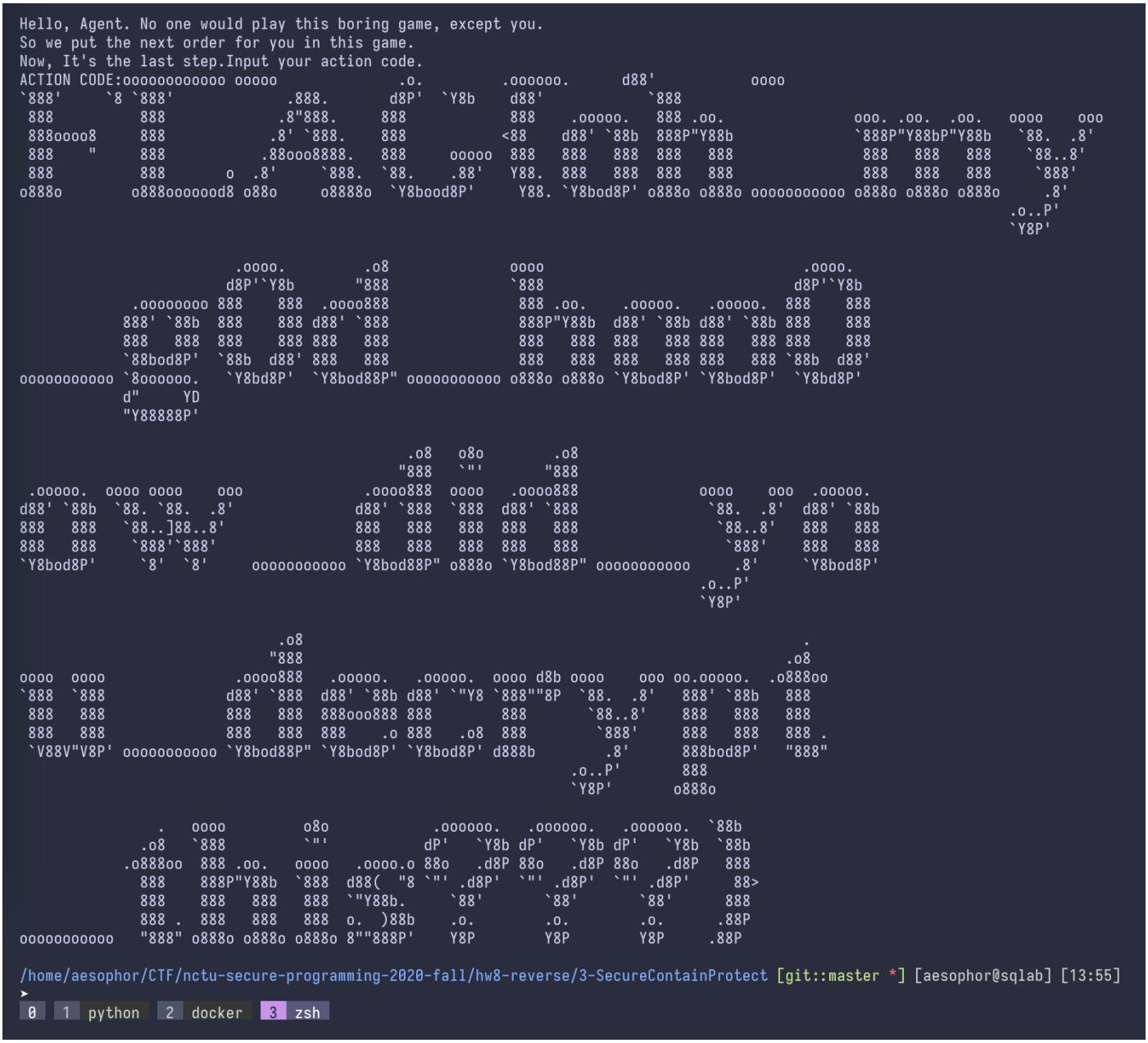
```
Key = 6e: edbsxqu^uid^*nbtld :.!(....c|{|.$d#<7>:
Key = 6f: decrypt_the_+ocume!;. ).....b}z}~%e"=6?;
Key = 70: {z|mfok@kwz@4p|jrz>$.?6....}beba:z=") $
```

0x6f(o) 有： `decrypt_the+ocume`

這邊很像 action code，經過網路上的 ASCII art generator 可以發現 Roman 這個字型是由大量的 0x20 與 0x6f 組成的



只要我們將第一行拿去 XOR 就可以拿到 action code： `decrypt_the_document_of_SCP-2521`



輸入該 action code 後即可獲得 flag： `FLAG{oh_my_g0d_hoo0ow_did_you_decrypt_this???}`