HW0x02 Writeup

trace

`trace` 裡面包著另一個執行檔 `cs_2022_fall_ouo`，執行 `trace` 後會先建立 `cs_2022_fall_ouo`，然後像個 debugger 一樣來逐步執行他，但是會發現沒辦法直接執行 `cs_2022_fall_ouo`。

這裡叫 `trace` 為 tracer，`cs_2022_fall_ouo` 為 tracee。

tracer 會不斷地看 tracee 的 memory space，如果某個位址的值為 `0xE8CBCCDEADBEEFE8` 就會替換成 `0x9090909090909090`。

知道這點後就手動去改 tracee 裡出現 5 處的 `0xE8CBCCDEADBEEFE8` 並替換成 `0x9090909090909090`，再去看他的 assembly code 就看到有一組資料做 XOR 後跟輸入做比較，還原他就是 flag 了。

```
unsigned char flag[] = {
  0x37, 0x3D, 0x30, 0x36, 0x0A, 0x25, 0x03, 0x30, 0x12, 0x42,
  0x2E, 0x3C, 0x42, 0x2E, 0x40, 0x37, 0x2E, 0x24, 0x2E, 0x12,
  0x30, 0x3F, 0x0C, 0x00
};

int main() {
    for(int i = 0; i < 24; i++) {
        flag[i] ^= 0x71;
    }

    printf("%s\n", flag);
}
```

pwn_myself

從題目敘述知道有用到 OpenSSL，並且跟鍵盤有關。

一開始用 IDA 只看到 main 只呼叫了一個 function 後就結束，完全沒用到其他一大堆 function，也許是 main 執行過程去計算了其他位址然後跳過去執行。

用 gdb trace 的確跑到了其他 function 裡，做了一大堆事：

1. 嘗試去 `/dev/input` 裡打開某個檔案，八成就是鍵盤。
2. 從鍵盤得到一組輸入。
3. 讓這輸入滿足些條件後進到另一個 function。
4. 把輸入搬到某個 buffer 裡。

```
unsigned __int64 sub_5555555BA43F()
{
  int dec_len; // [rsp+4h] [rbp-6Ch] BYREF
  unsigned int i; // [rsp+8h] [rbp-68h]
  int out_len; // [rsp+Ch] [rbp-64h]
  __int64 v4[11]; // [rsp+10h] [rbp-60h] BYREF
  unsigned __int64 v5; // [rsp+68h] [rbp-8h]

  v5 = __readfsqword(0x28u);
  out_len = Encrypt();
  for ( i = 0; i <= 0x2F && out_buf[i] == ciphertext[i]; ++i )
    ;
  if ( i == 48 )
  {
    Decrypt(congratulations, 0x20u, v4, &dec_len);
    SendData(v4, dec_len);
    exit(1);
  }
  SendData(out_buf, out_len);
  return v5 - __readfsqword(0x28u);
}
```

接著進到這個 function，繼續 trace 下去發現 code 的複雜度爆增，沒意外就是進到 OpenSSL 的 code 了，接著是找出對應的 source code。

我是用 function 裡出現的字串丟去 github 搜尋，用 assertion 的條件找到這是在 openssl/crypto/evp/evp_enc.c 裡的 function，也就是加解密的流程。

因為這種 library 都要遵從他們規定好的流程呼叫 API 才能正確執行，我就找了個範例參考，照著流程還原 code，找到 `key` 跟 `iv`。

最後是要知道用了什麼 cipher，這部分我還原不出來，但是有 `key` 有 `iv` 我就猜是 AES，而是什麼 mode 就從 CBC 開始試，結果解密的部分成功得到 `[*] Congratulations`，所以 flag 應該是在加密的部分。

```
from Crypto.Cipher import AES

key = bytes.fromhex("4B29470F38D4A34D1C9F4FC774E4296A")
iv = bytes.fromhex("B59AEC9251E25E3F9081E427192E5029")
ct =
bytes.fromhex("D2B240F2DE77E085FDE5BFB1EBF76418E4AD85EF8068DA2C252DE1F8DDE70B59E8D757372FB54125785AB982228D8126
")

cipher = AES.new(key, AES.MODE_CBC, iv=iv)
pt = cipher.decrypt(ct)

print(pt)
```

OOXX

用 IDA trace code 就會找到遊戲的核心邏輯，可以看到有個 array 紀錄版面資訊，然後我們 `O` 是對應到 1，`X` 是對應到 2，當 `O` 贏了就會顯示 flag。

直接改指令讓 `board[v5] = 1` (讓電腦下 `O`)，這樣怎麼玩都會贏了。

```
__int64 __fastcall sub_7FF6E8821C50(__int64 a1, __int64 a2, int x, int y)
{
  __int64 result; // rax
  int v5; // [rsp+38h] [rbp-10h]
  int pixel_idx; // [rsp+3Ch] [rbp-Ch]

  pixel_idx = x / 200 + 3 * (y / 200);
  result = pixel_idx;
  if ( !board[pixel_idx] )
  {
    board[pixel_idx] = 1;
    draw_circle(a1, 200 * (x / 200) + 20, 200 * (y / 200) + 20, 160, 160);
    if ( game_end() )
      return sub_7FF6E88249B0(a1, sub_7FF6E8821C50);
    v5 = opponent_turn();
    board[v5] = 2;
    draw_line(a1, 200 * (v5 % 3) + 20, 200 * (v5 / 3) + 20, 160, 160);
    draw_line(a1, 200 * (v5 % 3) + 180, 200 * (v5 / 3) + 20, -160, 160);
    result = game_end();
    if ( result )
      return sub_7FF6E88249B0(a1, sub_7FF6E8821C50);
  }
  return result;
}
```

---

trojan

有給一個執行檔跟一個 pcapng file，是 Wireshark 的封包紀錄。

一樣用 IDA trace code，除了跟 WinForm 初始化相關的 function 外有兩個部分，一個是 建立 socket 跟接收傳送訊息，另一個是在 while loop 裡的 function，剛好我寫過用 Windows API 做視窗擷取，用的 API 長得很像，但沒有指定要擷取的 process，所以應該是螢幕擷取。

跟 socket 相關的部分，有個 function 在等待接收訊息，等著接收到一組 key，收到後會回傳兩筆資料回去，但還不知道這兩筆資料是什麼，所以就寫個 socket program 配 x64dbg 看傳了什麼東西，結果有看到 png header，就知道是要傳送螢幕截圖，第一筆是 byte 數，第二筆是用 XOR 加密過的 png 檔。

大致了解執行檔的流程也沒發現什麼特別的東西後就想到這 pcapng file，應該是有張螢幕截圖記錄在裡面。把資料取出來解密就找到 flag 了。

```
#include <stdio.h>

#define BUFSIZE 0x025980

unsigned char key[] = "0vCh8RrvqkrbxN9Q7Ydx";
unsigned char buf[BUFSIZE];

int main() {
    FILE *r = fopen("./enc.png", "rb");
    FILE *w = fopen("./dec.png", "wb");

    fread(buf, BUFSIZE, 1, r);

    for(int i = 0; i < BUFSIZE; i++) {
        buf[i] ^= key[i % 21];
    }

    fwrite(buf, BUFSIZE, 1, w);

    fclose(r);
    fclose(w);

    return 0;
}
```

---

dropper

用 Detect It Easy 看到執行檔被 UPX 壓縮，就先對他解壓縮

```
.\upx.exe -d dropper.exe
```

接著在 `main` function 看到很多組加密過的資料以及一個 function 為他們解密。解密後就看到有 dll 檔跟 Windows 的 API，為一組資料做加密後寫入 Registry 裡。

最後就照著文件看懂 API 跟參數，照著執行一次就看到加密過後的資料正是 flag。

```
#pragma comment(lib, "crypt32.lib")

#include <stdio.h>
#include <Windows.h>
#include <Wincrypt.h>

unsigned char flag[] = {
  0xE1, 0xD6, 0x26, 0x81, 0x83, 0x9F, 0x36, 0x8F, 0xF6, 0x39,
  0x89, 0x7A, 0x92, 0x97, 0x84, 0xA4, 0x70, 0x04, 0x61, 0x4B,
  0x77, 0xED, 0x58, 0x5D, 0xE0, 0xE1, 0x75, 0x3A, 0xF9, 0x34
};

int main() {
        HCRYPTPROV phProv;
        HCRYPTHASH phHash;
        HCRYPTKEY phKey;
        BYTE data = 0xFF;
        BYTE *block;
        size_t size = 0x1E;

        CryptAcquireContextW(&phProv, 0i64, 0i64, 1i64, 0);
        CryptCreateHash(phProv, CALG_SHA, 0i64, 0i64, &phHash);
        CryptHashData(phHash, &data, 1i64, 0i64);
        CryptDeriveKey(phProv, 26625i64, phHash, 1i64, &phKey);
```

```
        CryptDestroyHash(phHash);

        block = (BYTE*)malloc(size);

        if (!block)
                exit(1);

        ZeroMemory(block, size);
        CopyMemory(block, flag, size);

        CryptEncrypt(phKey, 0i64, 1i64, 0i64, block, (DWORD*)&size, size);

        printf("%s\n", block);

        return 0;
}
```