計算機安全 HW8 Writeup

- Real name: 張瀚文
- Nickname on course website: Hwww
- Student ID: b07505027.

wishMachine

解法

雖然這題沒有debug資訊，但慢慢找可以找到幾個主要的函數，其中主要的檢查函數，慢慢讀的話可以發現，程式會將輸入進去的serial（長度為70），檢查 `f(serial[num2 + j]) == (num3或num4)` ， `f` 是函數，可能有五種：

1. 費波納契
2. 某種累加
3. 乘135
4. xor 1383424633
5. 0xFAC0B00C累減0x7788或0x78

根據已經存好的表決定f要用什麼函數

通過檢查後，會讓已經存好的flag ^= serial，重複1000個serial後就可解密為正確的flag，

上述的num2, num3, num4, flag, f都是存好的東西，可以從記憶體或stack撈出來

也就是說我們已經可以自行算出serial了，因為 `serial[num2 + j] = invf(num3或num4)` ，所以只要自己寫出五種函數相對應的反函數invf就可以得到flag了 ~

程式碼可參考 `code/wishMachine/` 底下的：

1. `wish.py` ：主程式
2. `fib` 跟 `fib.cpp` ：費波納契表。原本是用Python寫，但中間發現因為C會overflow，算出來的答案會跟Python不同，只好用C先建表
3. `facebook` 跟 `facebook.cpp` ：因為一直出問題，怕跟fib一樣所以用C寫一下表
4. `bur` , `flag` ：存num2, num3, num4, f, flag等資料

Curse

解法

根據提示（或自己慢慢猜），先找到原本在data中，被寫入的code是在哪裡被執行的，發現是在 `.text:00401A8E call eax` 這行，

而跟進去eax後（假設叫他 `bad()` 函數），因為ida不會解析得很漂亮，甚至要等ida跑完hex才會被解析成指令，或是要自己手動將他轉為指令，因此只能慢慢尋找程式的關鍵點，而且麻煩的是每次寫入的code的記憶體Base不一樣，如果忘記紀錄Base就可能要重來...但只要乖乖花時間跟著跑就會找到bad中最關鍵的地方，bad會在位移為13DE的地方呼叫位移1573的函數（假設叫 `real_main()` ），如 `006413DE call near ptr unk_641573`

找到 `real_main()` 後就輕鬆多了，研究一下後知道， `real_main()` 裡面包含了輸入、將輸入做某種轉換、一段疑似跟flag有關的奇怪的字串 `"^cm1;Fo]Zb..."` 、最後輸出一段裝飾用的文字。這裡把printable ascii丟進去測試一下輸入的轉換是什麼，結果發現這個轉是可以直接用輸入跟輸出建對照表的，然後根據其他題的解法，猜測輸入應該就是flag，flag會被程式轉換成 `"^cm1;Fo]Zb..."` ，所以建好表後把他反轉換就會得到flag拉～

程式碼可參考：

1. `code/Curse/solve.py`
2. `code/Curse/note` ：一些過程記錄

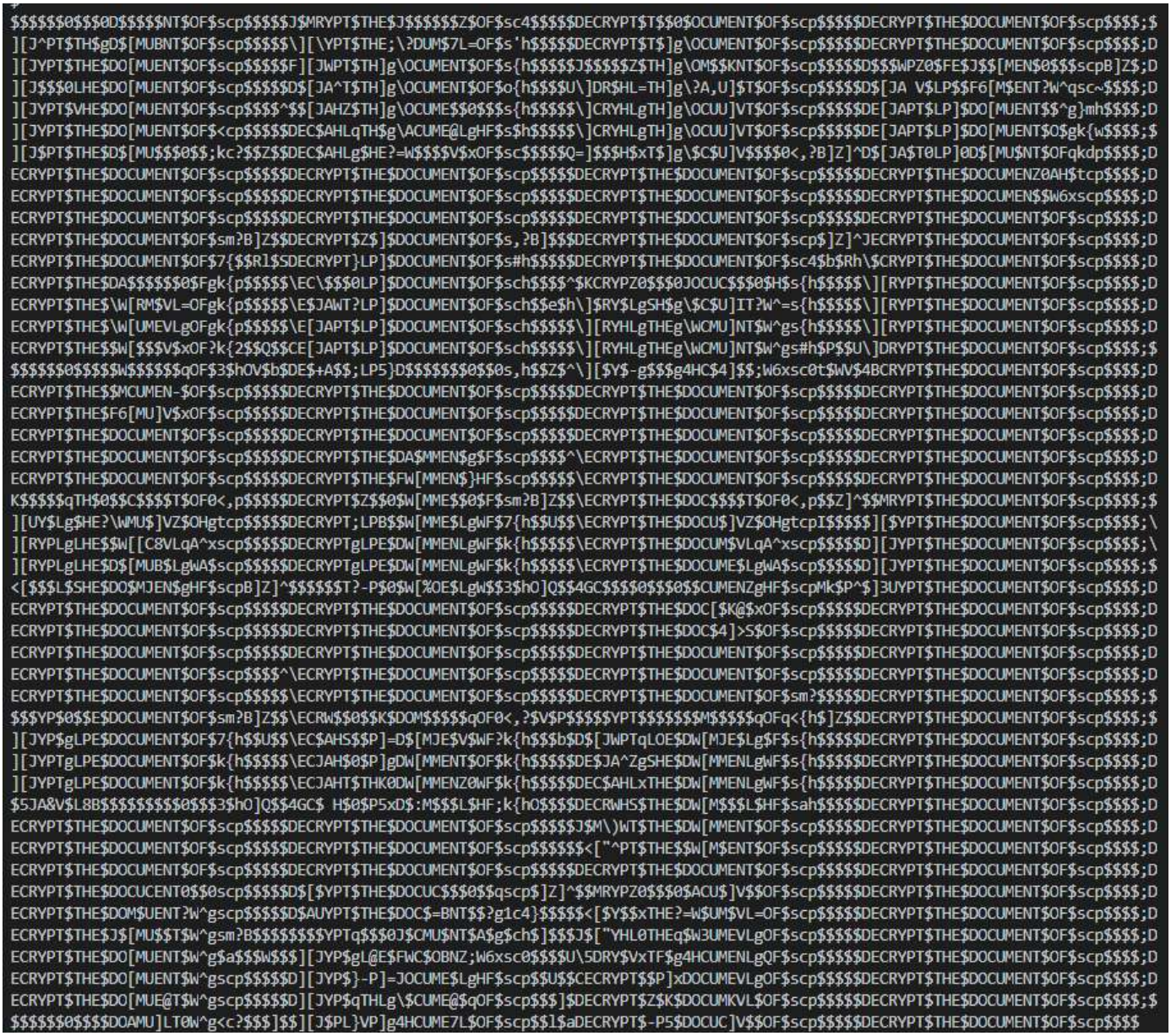
SecureContainProtect

解法

這題可以先解出數獨，存成一個陣列ans，送出finish後，需要輸入一個action_code，用ida看按下finish後會進的這個函數（假設叫finish函數）非常的簡單易懂，可以很明顯的看出finish函數會把兩個存好的陣列s1, s2分別跟ans做xor之後輸出，而這就是我們可以看到的TOP SECRET等字。

接下來就是重點部分，他會把一個存好的encoded_flag跟ans跟我們要輸入的action_code做xor（ `flag = encoded_flag ^ ans ^ action_code` ），然後檢查是否正確並輸出。其中encoded_flag跟ans是已知，所以我們可以先把它們做一次xor（變成 `flag = encoded_flag ^ action_code` ），也就是說剩下只要找到action_code這個密鑰就好。

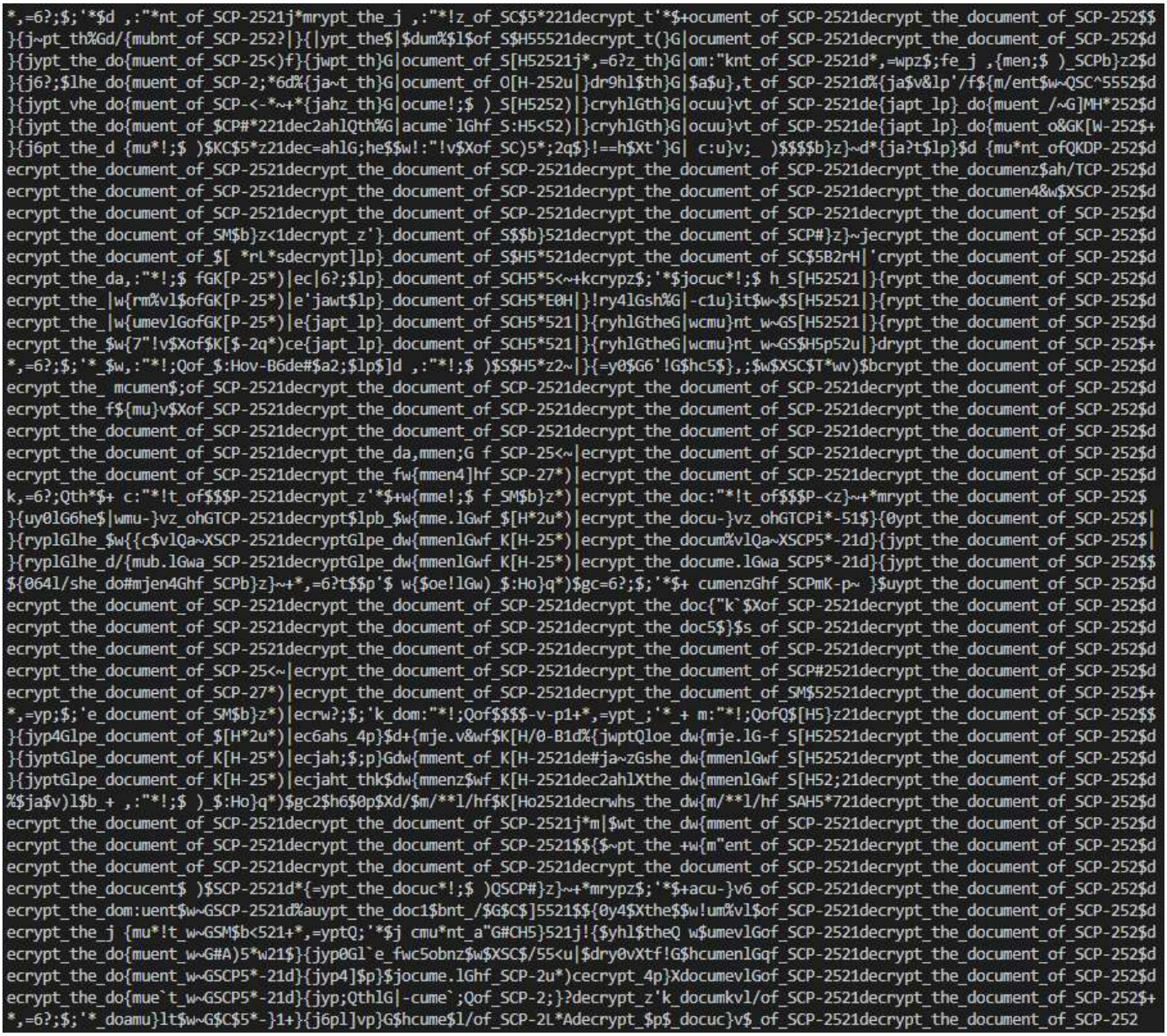
稍微觀察一下encoded_flag，可以發現貌似有個規律，大致是由「大值字元 * 26 + 小值字元 * 6」組成，共32個字（如下圖，$是一些換行之類的特殊字元），猜測action_code的長度應該跟32這個數字有關係，而這個數據也跟ida顯示的 `%39s` （action_code小於40個字）吻合。



這裡有一個簡單但有用的方法：猜測flag因為是由ascii art組成的，應該會有很多空白（已知明文），因為xor的性質：

```
" " = encoded_flag ^ action_code
encoded_flag ^ " " = action_code
```

所以如果幸運的話，就可以看到很多action_code散落在各處，實際上去比對encoded_flag真的有很多重複的序列。xor之後會有下圖：



發現其中有很多很明顯的可讀字串，action_code: `decrypt_the_document_of_SCP-2521` ，就可以解回flag了 ~

程式碼可參考 `code/SecureContainProtect/simply_find.py`

另解

因為程式最後會檢查 `sum(flag)` 有沒有等於257498，平均每個字元約43，而如果先把encoded_flag加總的話，會發現目前的sum非常大，所以先試著找到一個長度為w的action_code，讓對應的flag可以最小，再由這個action_code往上暴搜，其中action_code跟flag要是printable ascii。結果發現在w = 32的時候，action_code的每一項讓相對應的flag值最小的解就恰巧是答案了><也不需要往上暴搜 ~

程式碼可參考 `code/SecureContainProtect/find_action_code.py`