**HW4 Writeup**

**R10922043 黃政瑋**

**CTF account: cwhuang1937**

1. **sandbox**
   1. 分析題目

先用checksec會發現所有保護機制都打開了，但仔細分析source code後，會發現new\_code\_buf會mmap到0x40000上面，並有RWX的權限(後面會改成只有RX的權限)。在main的最後會執行我們寫入的shellcode，再加上這題的read並沒有BOF的問題，故往shellcode這個方向嘗試。

* 1. 分析monitor的行為

這邊寫一個簡單的shellcode來測試syscall與call <reg>的pattern會被怎麼處理。

一張含有 文字, 裝置, 儀錶 的圖片

自動產生的描述

從gdb可發現會替換成call r8(r8即monitor的address)，接著epilogue 會做exit的動作。因此可借用epilogue的syscall來拿shell。

一張含有 文字 的圖片

自動產生的描述

* 1. 寫payload

寫入execve("/bin/sh", 0, 0)所需的regs，並藉由r9這個reg塞入address，來跳過上述的mov rax, 0x3c，即可成功拿到shell。

一張含有 文字 的圖片

自動產生的描述

1. **fullchain-nerf**
   1. 分析題目

大致掃過source code後，發現有BOF與FSB的漏洞可以用，接著用checksec檢查後，由於有Partial RELRO故可用上課教的**從GOT來Leak libc**，方便拿到後面所需的ROP gadgets。

一張含有 文字 的圖片

自動產生的描述

* 1. Leak ELF與libc的base address

藉由FSB來leak stack上所存的\_\_libc\_csu\_init address，來反推 ELF的base address。

一張含有 文字 的圖片

自動產生的描述

第一次的ROP並搭配GOT來leak出put在libc的address，即可反推出libc的base address。

一張含有 文字 的圖片

自動產生的描述

* 1. Leak old rbp

這邊用FSB來leak出在function chal的rbp，以便後續做stack pivoting可以返回。**由於這邊一開始並沒有試著拿rbp的值，變成之後的stack會在global的bss section，結果stack寫到後來會溢出RW的範圍，變成最後會在只能R的區段寫入，而導致segmentation fault。**

**一張含有 文字 的圖片

自動產生的描述**

* 1. Stack pivoting到global上做ROP

將ROP寫入到global variable上，接著用stack pivoting跳過去，做完ROP後，將**原本的old rbp + 10(即再做下一次chal的位置)pop到rsp**，即可成功返回原本的stack並再重新一次chal function。

一張含有 文字 的圖片

自動產生的描述

重複三次這樣stack pivoting過去做ROP再返回原stack的過程，即可做完open、read、write，並成功拿到flag。以下為這三次ROP chain的payload。

一張含有 文字 的圖片

自動產生的描述

**一張含有 文字 的圖片

自動產生的描述**

**一張含有 文字 的圖片

自動產生的描述**

1. **fullchain**
   1. 分析題目

這題跟nerf那題很不一樣的是**不能直接用BOF打到ROP**，查看很久**只有FSB能打**，但FSB只能改**往下stack的pointer指向的值**，因此並不能直接去改cnt的值，而必須先寫入一個pointer到stack上，然後間接去改值，仔細不斷用gdb看stack後，**發現在rbp - 0x8 ~rbp-0x10(16$)的位置可以藏東西**，且每次myread可以讀24個bytes，而local variable的size只有16個bytes，因此這8bytes的空間是可以拿來當跳板做FSB，而不會被影響到。

一張含有 文字 的圖片

自動產生的描述

* 1. 改cnt的值

事實上這邊卡了非常久，一直想不到怎樣在有限的3次內將cnt改非常大，前面試了非常多的方法至少都需要四次，後來**跟R10922046(李柏漢)討論**，他給了我一個很關鍵的提示!因為scanf那邊是%10，且strncmp只有比較前5個字，因此**write????是可以通過的，因此這邊就能做一次FSB**，可以省下許多次數。

一張含有 文字 的圖片

自動產生的描述

如圖所示，第一次讀出來stack\_base，第二次利用那個hole放入我們要存cnt的地址，第三次即可成功將值改成5，此時再用一次的手法即可將cnt改成非常大，方便後續做事情。

* 1. Leak出elf與libc的base address

藉由FSB來leak stack上所存的\_\_libc\_csu\_init address，來反推 ELF的base address。

Libc這邊則用GOT hajack的方式將printf\_got**以%s的方式印出來**。由於這邊在連remote很容易拿錯address，因此後面改成故意先走一次myset，並觸發puts("Too more")，使puts被解析進GOT表後，即可成功將libc的base address leak出來。

一張含有 文字 的圖片

自動產生的描述

* 1. 寫FSB改任意address的function

由於這題寫到後面會發現，從頭到尾都是利用那個hole並利用FSB寫入我們想要的值，因此把寫入address的動作包成function，以便後續使用。

另外，由於最多一次只能寫到%hn的大小(2bytes，若寫到%n則會失敗)，且address的前4個bits都為0，因此**只要分批來寫入三次，並對target address做相對應的平移，即可寫入任意值到我們想寫的address**。而單純寫value的話，由於都不會太大，因此只要寫一次即可。

一張含有 文字 的圖片

自動產生的描述

一張含有 文字, 螢幕 的圖片

自動產生的描述

* 1. 定義ROP gadget並做stack pivoting

這邊用上述定義好的function，**將stack pivoting後的位置放到stack下方的0x20的地方**(原本是放在global variable上，但不知道為何remote太容易出現EOF了，改成stack上有稍微改善)，**接著將chal的return address改成leave\_ret，且exit@got也改成leave\_ret，後續會將cnt歸0，使其觸發leave\_ret**，最終成功跳到stack下方的0x28處執行ROP chain，即可成功做到open、read、write來拿到FLAG。

一張含有 文字 的圖片

自動產生的描述

不確定是不是寫法問題，但我**EOF的機會非常非常高**，一直無法成功拿到FLAG，後來**換到系館的網路後，成功率明顯高非常多**，最終也成功解出來了!!!

一張含有 文字 的圖片

自動產生的描述

1. **final**
   1. leak libc base address

已知unsorded bin 為**Circular Doubly Linked List**，當只有一塊chunk進入該bin時，**fd與bk皆同時指向main\_arena**，因此可從此來leak出libc的base address。首先先malloc一塊0x410大小的chunk，再malloc一塊chunk來避免當0x410的chunk被free時會觸發consolidation。接著將該0x410的chunk free掉後再malloc同大小的回來，**此時bk所指向的address並不會被洗掉**，因此可從這邊leak出libc的base address。

* 1. leak heap address

因為這題的animal release後，並沒有把animal的pointer設為null，因此會有UAF的漏洞。此時可以先malloc兩塊一樣大小的chunk，再free掉，此時後面free掉的那個chunk的fd會指向第一塊chunk，因此用play即可成功leak出heap address。

* 1. UAF to get shell

由前面拿heap address的過程中，兩個0x30的chunk在heap tcache裡面。因此這邊先malloc一塊chunk 1出來，其中的name再malloc一塊同樣大小的chunk 0，即原heap中的另外一塊0x30的chunk，**因此這邊寫入chunk 1的name，即是寫入到chunk 0上**，這邊將bark的fptr蓋成system，type蓋成’/bin/sh\x00’，即可從UAF的漏洞成功get shell。

* 1. One gadget to get shell

如同前面(3)的做法，一樣在chunk 1裡面的name malloc一塊一樣0x30大小的chunk 0，**此時蓋寫len為一大數，方便後續寫入來製作one\_gadget，而name的address即蓋寫成等等觸發one\_gadget的chunk 2的address**(0xbe0)。**接著由UAF，可對chunk 0的name改值，即是chunk 2的data任意寫入**，此時將type與name設為NULL(rdi、rsi皆設為NULL)，並將fptr改成one\_gedget的address。此時play即可去觸發one\_gadget，但由於這題沒有符合條件的one\_gadget，故無法用這個方法get shell。

一張含有 文字 的圖片

自動產生的描述

* 1. Tcache poisoning to get shell

如同(4)，一樣在chunk 1裡面的name malloc一塊一樣0x30大小的chunk 0，而該chunk 0 的name指向後續的chunk 2。**藉由UAF，可對chunk 0的name改值，即是chunk 2的data任意寫入**。這邊要做double free，要繞過偵測，故以free(A)->free(B)->free(A)的方式繞過，如下圖所示。

一張含有 文字 的圖片

自動產生的描述

接著將上圖b90的位置蓋寫成\_\_free\_hook – 8，下次buy的時候即可成功跳到上面，並將’/bin/sh\x00’放到\_\_free\_hook – 8的地方，而system的address放到free\_hook，當下次執行free的時候，即可成功跳到system上，並把chunk的內容當作參數(\_\_free\_hook – 8的/bin/sh)，此時即可get shell。一張含有 文字, 時鐘, 裝置 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

1. **easyheap**
   1. 分析題目

看完source code會發現，在delete的那邊並沒有將對應的ptr設為NULL，搭配edit即有**UAF**可以利用。這題寫入string與final那題不同，每次讀進來的string都會被加入\0，因此無法像final那樣直接從unsorted bin leak出libc base address，因此**這題先從leak heap base address下手，接著再用UAF來間接拿到libc。**

* 1. Leak heap base address

Chunk進入tcache後產生的**key，剛好會與index的位置相同**，所以此時list\_book的時候，會將該book的key以整數印出來，即可成功反推出heap base address。

* 1. Leak libc base address

add一個0x420的chunk，再從unsorted bin add回來，可從他的bk拿到main\_arena的address，可從這反推出libc base address，但因為寫入string時結尾會被強制加上\0，不能直接將name寫滿後跟著leak出來。如下圖index所示，這邊利用UAF，使得**chunk 2 的name 所用的chunk會與前面add過的chunk 0相同，因此可以從chunk 2間接寫入bk的位置**，接著從chunk 0的name印出來即可拿到libc base address。，

一張含有 文字 的圖片

自動產生的描述

* 1. Use tcache poisoning to get shell

這邊先將前面還在free的chunk皆add出來，方便後續計算offset。接著用與拿libc相似的手法，如下圖index所示，首先將chunk 9 free到tcache裡面，因為chunk 8的name所用的chunk與chunk 7相同，故這邊可以**從chunk 8修改chunk 7 的內容，再從chunk 7 間接改chunk 9 的內容**，並再free一次chunk 9即可做到**double free**。接著將chunk 9的內容改成free\_hook-8後，再一次add即可成功**讓tcache指向free\_hook-8的地方，並將hook改成system，參數即/bin/sh**，此時再一次free，即可成功get shell。

一張含有 桌 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述 一張含有 文字 的圖片

自動產生的描述

1. **FILE note**
   1. 分析題目

看完source code後，會發現這題的gets有BOF的漏洞可以打，並可以從note\_buf來蓋到file structrue。由於這題並沒有提供libc或heap的address，因此需要先leak出來。

如圖所示，可以從offset 2a0藉由BOF蓋到4b0的內容，因此可以hijack file structure。

一張含有 文字 的圖片

自動產生的描述

* 1. leak libc base address

由於並不知道heap的address，因此只能試著從BOF改file structure來試著leak出一些information。因為要leak出東西的話一定要將\_fileno改成1或2，才能輸出到畫面上來recv，因此這邊**第一次的fwrite先試著將\_fileno改成1**，接著會發現file structrue上buf的pointer都會有殘留值。

後來這邊卡關了很久，後來參考南瓜大大的筆記並看glibc的source code。

一張含有 文字 的圖片

自動產生的描述

從圖中可以發現藉由**將flag的0x1000設起來，就可以讓IO\_read\_end與\_IO\_write\_base不一樣**，此時就可以**直接將IO\_read\_end蓋成0，並將\_IO\_write\_base的最後一個byte蓋成\x00，使得fwrite會將\_IO\_write\_base到 \_IO\_write\_end的data給leak出來**，也就是如下圖offset 600~690的位置，而在680的地方剛好會有\_IO\_wfile\_jumps的address，可以從這個反推出libc base address。

一張含有 文字 的圖片

自動產生的描述

* 1. 處理remote拿不到libc base address的坑

Local成功拿到後，remote卻無法拿到，這邊試著leak完後傳一些trash來導致crash，會發現其實是有leak出來的，而且buf明顯比local還要多，**這邊的解法是當要連到remote時，在leak後重複做fwrite 7次，試著將remote的buf擠出來**，即可成功拿到libc base address。

後來有跟**R10922046(李柏漢)討論**，我們試著leak出fwrite時allocate出來的buf size，可以發現**local是0x400，remote是0x1000，而我們每次寫入的內容為0x200**，所以remote第7+1次才會噴出來我們猜是因為這樣，但為何會有這個落差，我們推測是**\_IO\_file\_doallocate這邊的判斷tty有關**，如下圖所示。

一張含有 文字 的圖片

自動產生的描述

但為何local寫第一次就噴出來，而不是第二次呢，這邊我們討論glibc src code很久仍沒有很懂，只能**推測是下圖的這個地方有觸發flush，以至於local第一次就會leak出來**。

一張含有 文字 的圖片

自動產生的描述

* 1. one\_gadget

由於這題看起來只有開shell這個洞才能拿到flag，變成說只能藉由fwrite來達到任意寫才有辦法，這邊也是參考南瓜大大的筆記後搭配看glibc source code，才想出來怎麼解。

一張含有 文字 的圖片

自動產生的描述

如上圖所示，**\_IO\_new\_file\_xsputn中會判斷\_IO\_write\_ptr到\_IO\_write\_end是否有空間，若有的話會將該空間大小的data給寫入**。

又已知fwrite在底層會call到\_IO\_file\_overflow(\_IO\_file\_jumps + 0x18)，因此這邊**將\_IO\_file\_overflow hijack成one\_gadget**，並發現當要**執行one\_gadget時r15有滿足條件，r12會指向one\_gadget後面8bytes的地方，因此這邊在one\_gadget的後面寫入一個p64(0)**，即可成功藉由one\_gadget來get shell。

一張含有 文字 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述



Reference:

南瓜大大的筆記: [PWN cheatsheet - HackMD](https://hackmd.io/@u1f383/pwn-cheatsheet#FILE-exploit)

1. **beeftalk**

**解完其他題目已經確定破1000了，加上期末太忙這題就沒有解了。**