CTF Hw1 writeup

b05902127 劉俊緯, user ID: a127000555

Shortshell

FLAG{r34d_4RE_th3_es1esT}

- 利用內部鑲嵌read的指令就可以再read一次，但是size就可以自己調了。
- 雖然直接objdump會不知道到底size會限制多少，但是我們大致上可以利用read syscall的性質(例如這題的fd是stdin, 0)，的ax/di/si/dx來知道大概是這三行
  -
    ```
    mov     edx,0xa
    mov     rsi,rax
    mov     edi,0x0
    ```
  - 所以大概只能寫10個字。
- 此時為了利用最少的字，要利用在asm裡面其他人的值來代替本來我們用來設定參數的mov。
  - 此時看到rdi數字夠大，所以遷來rdx用，不然rdx直接mov就會多4個char。
  - rsi選擇不變，此時就第二次read就可以蓋過第一次的syscall
  -
    ```
    mov rdx, rdi
    xor rdi, rdi
    syscall
    ```
  - 最後造出來的字串長度是8。
- 之後就用一般的shellcode打法就好，但是前面必須加上第一次code的長度的nop來讓program counter下一個的指令就是我們第二次的code。
- code: `python3 shortshell.py` 就可以直接看到flag。

Shellsort

FLAG{SUPERB_sh311c0d1ng_ski115!!!}

- Keypoint: xor dword ptr [starting_point + offset] XOR_STR = `83 starting offset XOR_STR`
- 因為每個字彙被排序過，所以我們會想辦法過。
- 接下來就是尋找適當的4個有順序的指令， 可以依照一些性質：例如四個指令都差一個bit，這樣再xor後會比較好想。
- 要讓rsi的數字大，卻又不能靠mov(mov太肥太難掌控)，因此就有查到一個指令pushf，其hex夠大，而且在此題eflag為0x246，因此之後只要用pop rdx就可以讓rdx有個不錯的數字可以讀入(count)。
- 我們又要把指標給rsi，因此這樣轉讓我們可以用成 `push rdx pop rsi`
- 因此我們已經有了三個順序的指令：`push rdx pop rsi pop rdx`

- 之後只要給出一個number，使得這個number ^ 以上三個指令會在syscall以上，0x70以下，此外這個number也必須是小於0x70
- 找到了有了這麼個數字是0x6a後，在給一個垃圾指令使得順序不會亂掉，解譯又不會錯，因此我們找到了pop rcx。
- 如果是pop指令，我們就可以在更前面在多pushf就可以使的順序是好的。
- 最後在因為rdx+70的原因補個padding後就可以達成read的指令。

- 接下來做的事情就跟第一題一樣，丟出原本的指令+接下來的shellcode，就可以shellcode成功。
- code: `python shellsort.py` 就可以直接看到flag。