# Format String

2020-03-31

LYS

# What Wrong?

```c
#include <stdio.h>
int main(){
    char buf[50];
    scanf("%49s",buf);
    printf(buf);
    return 0;
}
```

# What Wrong?

```c
#include <stdio.h>
int main(){
    char buf[50];
    scanf("%49s",buf);
    printf(buf);
    return 0;
}
```

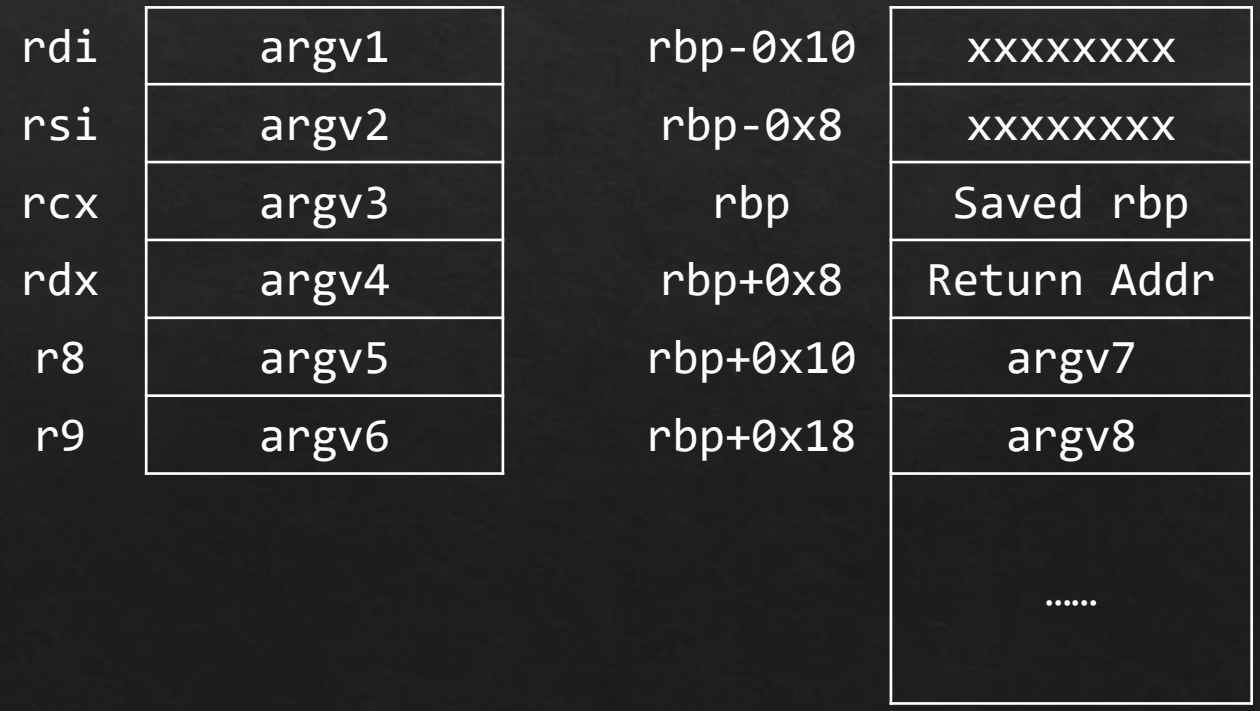
```
%p%p%p%p%p%p%p
0x7f0eec7f49f00x1(nil)(nil)(nil)0x702570257025702502570257025
```

# Format String Attack

- 當我們可以控制 Format String 的時候，就有機會做到任意讀寫
- 幾個關鍵的 Format
  - %p leak stack
  - %s leak 任意位置
  - %n write

# X64 Call function argument

- 前 6 個參數依序放在 rdi, rsi, rcx, rdx, r8, r9
- 剩下的參數放在 stack



# X64 Call function argument

- 前 6 個參數依序放在 rdi, rsi, rcx, rdx, r8, r9
- 剩下的參數放在 stack

| | |
|---|---|
| rdi | argv1 |
| rsi | argv2 |
| rcx | argv3 |
| rdx | argv4 |
| r8 | argv5 |
| r9 | argv6 |

| | |
|---|---|
| rbp-0x10 | xxxxxxxx |
| rbp-0x8 | xxxxxxxx |
| rbp | Saved rbp |
| rbp+0x8 | Return Addr |
| rbp+0x10 | argv7 |
| rbp+0x18 | argv8 |
| | …… |

- %p,%p,%p,%p,%p,%p,%p,%p
- argv2,argv3,argv4,argv5,argv6,argv7,argv8

# %p

- 將該參數的內容以 8byte 的 hex 印出來
- 通常用來 leak stack 上面的值

# %s

- 將該參數所存的指標的指到的 pointer 當成字串印出來
- 可以用來任意讀

# Lab1

- https://bamboofox.cs.nctu.edu.tw/courses/9/challenges/189

# %n

- 可以用來寫入值到該參數存的 pointer 指向的 memory
- 寫入的值是到目前為止印出的字元各數
  - E.X aaaaa%n
  - 這樣會寫入 5
- 可以指定寫入的長度
- %xc可以直接印出 x 個空白字元
  - E.X %8c%n
  - 會寫入 8

| | |
|---|---|
| %lln | 8 bytes |
| %n | 4 bytes |
| %hn | 2 bytes |
| %hhn | 1 bytes |

# %n

- 由於要寫入的值可能很大，如果要一次寫完會花非常多時間，所以通常會分次寫
- 看情況通常會一次寫 2 bytes 或 4 bytes
- E.X 想寫入 0xdeadbeef
  - 如果直接 %3735928559c%11n 會跑非常非常久，跑不完
  - 這時可以切兩段寫，切成 0xdead 跟 0xbeef
  - 從比較小的值先寫
  - %48879c%n%8126c%n
  - 8126 = 0xdead-0xbeef

# Lab2

- https://bamboofox.cs.nctu.edu.tw/courses/9/challenges/190

# Format String Chain

- 在某些情況下，可能沒辦法像前面一樣直接把要寫入的 Address 寫在 Format String 後面，然後用 %x$n 去寫入。
  - 例如你的輸入是在 .bss 或 .data，而且有 PIE 保護，那在 Format String 上根本碰不到那邊。
- 如果遇到這種情況，就可以用這個技巧
- 常見的有
  - RBP Chain
  - Argv Chain
- 這邊來介紹 RBP Chain

# RBP Chain

- func3 saved rbp -> func2 saved rbp
- func2 saved rbp -> func1 saved rbp
- 透過 %n func3 saved rbp 可以對 func2 saved rbp進行寫入
- 透過 %n func2 saved rbp 可以對 func1 saved rbp進行寫入
- 假如我們想寫入 0x12345678 到 0xdeadbeef
  - 我們可以透過 %n func3 saved rbp 把 0xdeadbeef 寫到 func2 saved rbp
  - 此時 func2 saved rbp 的值就會是我們要寫的目的地 0xdeadbeef
  - 接著透過 %n func2 saved rbp 就可以把 0x12345678 寫入 0xdeadbeef
- 假設在無法 leak stack address 又無法 stack overflow 的情況下，我們甚至可以利用 partial overwrite saved rbp 來寫掉 return address (因為位置只差 0x8 但是會需要一點機率)