台科資安研究社

Pwn 101

Inndy / inndy.tw@gmail.com

課前身家調查

- 有聽過我的課
- 會 C 語言
- 會使用指標，寫出來的程式不會 crash
- 知道什麼是逆向工程
- 認識任何一種CPU架構的常用指令集
- 會一點 x86 的組合語言
- 有玩過 Pwn

What is "Pwn"?

- 挖掘並且利用漏洞，取得主機控制權限
- 最基本的漏洞成因
  - 對於不可信任的輸入資料沒有進行妥善檢查
  - 越界讀寫
  - 邏輯漏洞
  - 暗門 / 後門

Pwnable -- Binary Exploit

- 對目標程式進行逆向，理解程式流程
- 找出漏洞、弱點
- 想辦法找出漏洞利用方式
- 撰寫攻擊程式，製造攻擊 payload

常見漏洞

- Memory Corruption
- Buffer overflow
  - Stack-based
  - Heap-based
- Integer overflow
- Race Condition

經典 Stackoverflow

```
int main() {
   ...
   printf("your name:");
   scanf("%s", name);
   ...
   return 0;
}
```

```
(low address)
AAAAAAAA <--- esp
00000000
00000000 <---      buffer
00000000
00000000
00000000
00000000
0083af3c <---      stack canary
ffeeddcc <--- ebp, saved ebp
00401234 <---      return address
abcdabcd
(high address)
```

經典 Stackoverflow

```
int main() {
   ...
   printf("your name:");
   scanf("%s", name);
   ...
   return 0;
}
```

```
(low address)
AAAAAAAA <--- esp
00000000
41414141 <---      buffer
41414141
41414141
41414141
41414141 <---      stack canary
41414141 <--- ebp, saved ebp
41414141 <---      return address
41414141
(high address)
```

經典 Stackoverflow

```
int main() {
   ...
   printf("your name:");
   scanf("%s", name);
   ...
   return 0;
}
```

```
(low address)
AAAAAAAA <--- esp
00000000
41414141 <---      buffer
41414141
41414141
41414141
41414141
41414141
41414141 <---      stack canary
41414141 <--- ebp, saved ebp
41414141 <---      return address
41414141
(high address)
```

*** Stack smashing detected ***

經典 Stackoverflow

- name 在堆疊上
- data 在靜態的位址
- scanf("%s", name);
  - 造成 buffer overflow
- 知道 stack 是怎麼運作的很重要

Stack overflow 攻擊技巧

- Return to Stack
  - 把 shellcode 寫在 stack 上，return 到 stack 上執行
  - 找到 jmp esp
- Return to libc
  - 用 ROP 執行 libc 裡面的 function
- Return to dlresolve
  - 利用 Dynamic Linking ，讓他直接帶我們去 system()

Stack overflow 攻防戰

- Buffer overflow, return to stack
  - Data Execution Prevention，讓資料不可執行
- Return Oriented Programming （ROP輕鬆談 by Lays）
  - Stack canary 檢查有沒有 overflow
    - ASLR 讓你猜不中
    - Shadow Stack，用另外一個 stack 存 return address
    - ROP is dead！
- 延伸閱讀：Jump Oriented Programming

Stack Frame

- esp 指向 stack 最後一個被 push 的值
- ebp 指向 stack frame 開頭
  - 用 ebp 更容易找到參數和區域變數
  - 實際範例：

```
int add(int x, int y)
{
   return x + y;
}
```

Stack Frame

```
0804848d <add>:
 804848d:    push   ebp
 804848e:    mov    ebp,esp
 8048490:    mov    eax,DWORD PTR [ebp+0xc]
 8048493:    mov    edx,DWORD PTR [ebp+0x8]
 8048496:    add    eax,edx
 8048498:    pop    ebp
 8048499:    ret
```

Stack Frame

- 以常見的 stack frame 構造來說
  - [ebp] => 上一個 ebp
  - [ebp+4] => ret address
  - [ebp+8] => arg1
  - [ebp+0xc] => arg2
  - [ebp+0x10] => arg3

例題：array

- 漏洞：陣列越界任意讀寫
- 陣列本體存在 stack 上面
- 可以直接修改 return address
- 已知 index = 20 存放 return address
  - 怎麼知道的？
    - 暴力 try
    - 從 stack offset 計算

例題：array

```
scanf("%d", &idx);

8048684: lea     eax,[ebp-0x58]
8048687: push    eax
8048688: lea     eax,[ebx-0x16c4]
804868e: push    eax
804868f: call    8048470 <__isoc99_scanf@plt>
```

例題：array

```
printf("Array[%d] = %d\n", idx, array[idx]);

8048697: mov     eax,DWORD PTR [ebp-0x58] // idx
804869a: mov     edx,DWORD PTR [ebp-0x4c+eax*4]
804869e: mov     eax,DWORD PTR [ebp-0x58]
80486a1: sub     esp,0x4
80486a4: push    edx // arg3
80486a5: push    eax // arg2
80486a6: lea     eax,[ebx-0x16a8]
80486ac: push    eax // arg1
80486ad: call    8048420 <printf@plt>
```

例題：array

```
Proof:

Your choice: 1
Read
Index ( 0 ~ 15 ) = 20
Array[20] = 134514765 // 0x804884d

---

8048848: call   80485da <chall>
804884d: mov    eax,0x0 // return address
```

pwntools

```python
from pwn import *

# io = remote('127.0.0.1', 9999) # TCP conn
io = process('/bin/cat')

# send(bytes); recv(bytes_count)
# sendline(bytes); recvline()
# recvuntil('Input :')
io.sendline('Hello')
r = io.recvline()
print(r)

io.interactive()
```

pwntools

```
from pwn import *

x = p32(0xaabbccdd) + p8(0x99) # p16
print(hexdump(x))
```

```
v = u32(x[:4]) # u8, u16, u32
print(hex(v))
```

例題：rop

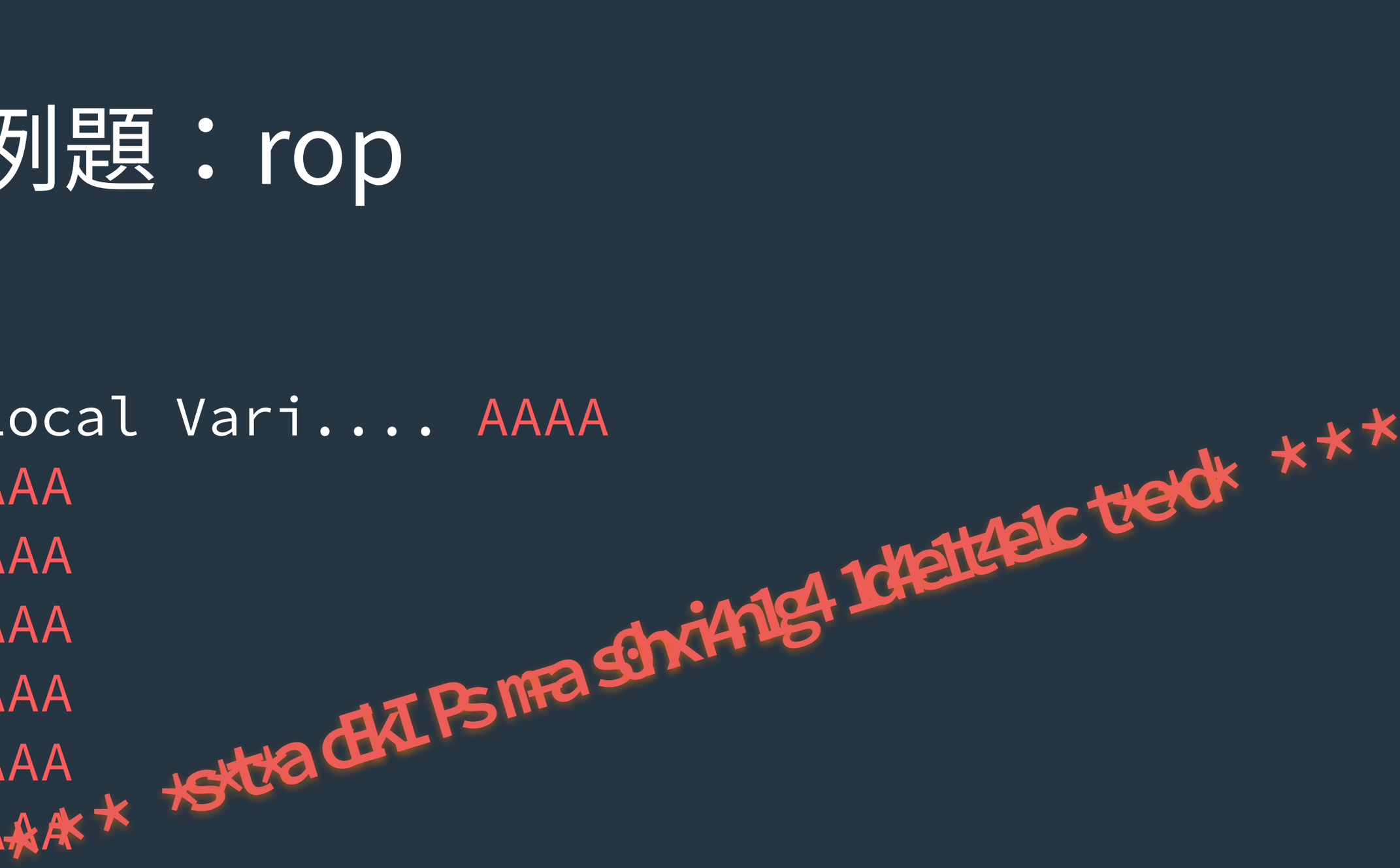
```
void overflow()
{
    char buff[32];
    printf("What's your name? ");
    gets(buff);
    strcpy(static_buffer, buff);
    printf("Hello, %s\n", buff);
}
```

例題：rop

```
[Local Variable]
[StackCanary] // if have
[OldEbp]
[RetAddress]
[Arg1]
[Arg2]
[Arg3]
```

例題：rop

```
[Local Vari.... AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
```



例題：rop

```
[System]
[Ret Addr for System]
[Arg1 for System]
```

Execution & Dynamic Linking

- 要講的東西有點多，這裡參考 angelboy 的簡報

資料洩漏利用方式

- 任意位址洩漏
  - 讀 GOT entry 拿到外部的位址
- 基於 stack 的洩漏
  - 讀 return address 拿到 libc 相對位址
- 找到 library 載入的位址，對於同一個 library 而言
  - function 對於 lib address 的距離是固定的
  - function 之間的相對距離是固定的

例題：leak

- 直接告訴你 stack 的 address
- 洩漏任意 address 的內容
- Buffer overflow
- 沒有 system()
- 沒有 "/bin/sh"

例題：leak

- 用 leak 功能找到 libc 位址
- 找到 system() 位址
- 把 "/bin/sh" 寫在 stack 上面
- system("/bin/sh");

例題：leak

ESP	ret
ESP+4	ret
ESP+8	ret
ESP+C	system
ESP+10	"AAAA"
ESP+14	ESP+18
ESP+18	"/bin"
ESP+1C	"/sh\0"

例題：rop2

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    setbuf(stdin, NULL);
    setbuf(stdout, NULL);

    puts("Just a simple overflow! Exploit me.");
    char buff[1];
    gets(buff);
}
```

例題：rop2

- 沒有 system()
- 沒有 "/bin/sh"
- 沒有任何功能，一開始就 buffer overflow

例題：rop2

- 透過 ROP 執行：
  - puts(puts@GOT); // 拿到 puts 的 address
  - main(); // 或是回到 _start
  - 再一次 ROP 的機會！
  - 透過 puts 已知的 address，算出 libc 位址
  - "/bin/sh" 和 system() 都存在 libc 裡面
  - system("/bin/sh");