

Pwn

[ELF 執行機制](#)
[Lazy binding 機制](#)
[Bookmark](#)
[C++](#)
[Exploit 方法論](#)
 [漏洞分析 SOP](#)
[前置知識](#)
[問題](#)
[關於漏洞發生的地點](#)
 [分析現有 patch](#)
[分析工具](#)
 [r2 找尋不安全的函數](#)
[pwntool 使用](#)
[z3solver](#)
[保護機制與繞過](#)
[Canary](#)
 [canary exploit](#)
[Stack pivot](#)
 [stack pivoting](#)
[洩漏 stack](#)
[ASLR vs PIE](#)
[PIE](#)
[ret2main](#)
 [RELRO \(RELocation Read Only\)](#)
[ROP 返回導向編程 \(Shacham\)](#)
[ROP 工具使用](#)
[linux 通用 gadgets](#)
 [__libc_csu_init\(\)](#)
[One Gadget](#)
[格式化字串 fmtstr](#)
[格式化字串應用情景](#)
[逆向工具](#)
[一些逆向的技巧](#)
[windbg](#)
[x64dbg](#)
[gdb](#)
[/bin/sh](#)
[修改 lib 搜尋路徑](#)
[GOT 蓋寫技巧](#)
[RELRO](#)
[緩解機制 Seccomp](#)
[ELF 修改保護](#)
[ELF](#)
[exit](#)
[pwndocker](#)
[Scanf trick](#)
[一些關於 pwntool](#)
[Heap exploit](#)
[UAF](#)
 [UAF 利用方法](#)
[forging chunk 構造假的 free chunk](#)
[Unlink](#)
[glibc Heap](#)
[tcache](#)

[HAL heap](#)
[Shellcode 編碼](#)
[Shellcode 轉可見字元](#)
[arm](#)
[aarch64](#)
[Linux](#)
[linux kernel](#)
[IOS kernel](#)
[windows kernel](#)
[Windows PEB](#)
[Thread](#)
[File struct](#)
[vtable](#)
[CET](#)
[對齊](#)
[GoLang](#)

angr 要注意 64 和 32 使用的暫存器 rbp ebp

環境變數

/proc/self/envIRON

/proc/self/cmdline

CWE-125: Out-of-bounds Read (4.2) - CWE

如只能蓋寫最後一 byte，是否可以洩漏一些字串，或 one gadget exploit (這不是 one gadgets

- 相關議程
- <https://www.youtube.com/watch?v=L9maBmiJGAc>
- https://hitcon.org/2017/CMT/slide-files/d1_s1_r0.pdf

然而 .bss 應該是可寫的 0x804b000 到 0x804c000 應該都是可以使用的最好找為 0 的況

ShiftLeft Ocular 可以挖掘內存相關的漏洞

<https://ocular.shiftright.io>

緩衝區蓋寫：當前函數的ret addr

任意地址蓋寫：蓋寫掉執行任意地址蓋寫的 retaddr, 如 read, fgets 可控 buffer 位置的時候

有時可能需要輸入無號數

```
import ctypes
>>> ctypes.c_ulong(-1)
```

ELF 執行機制

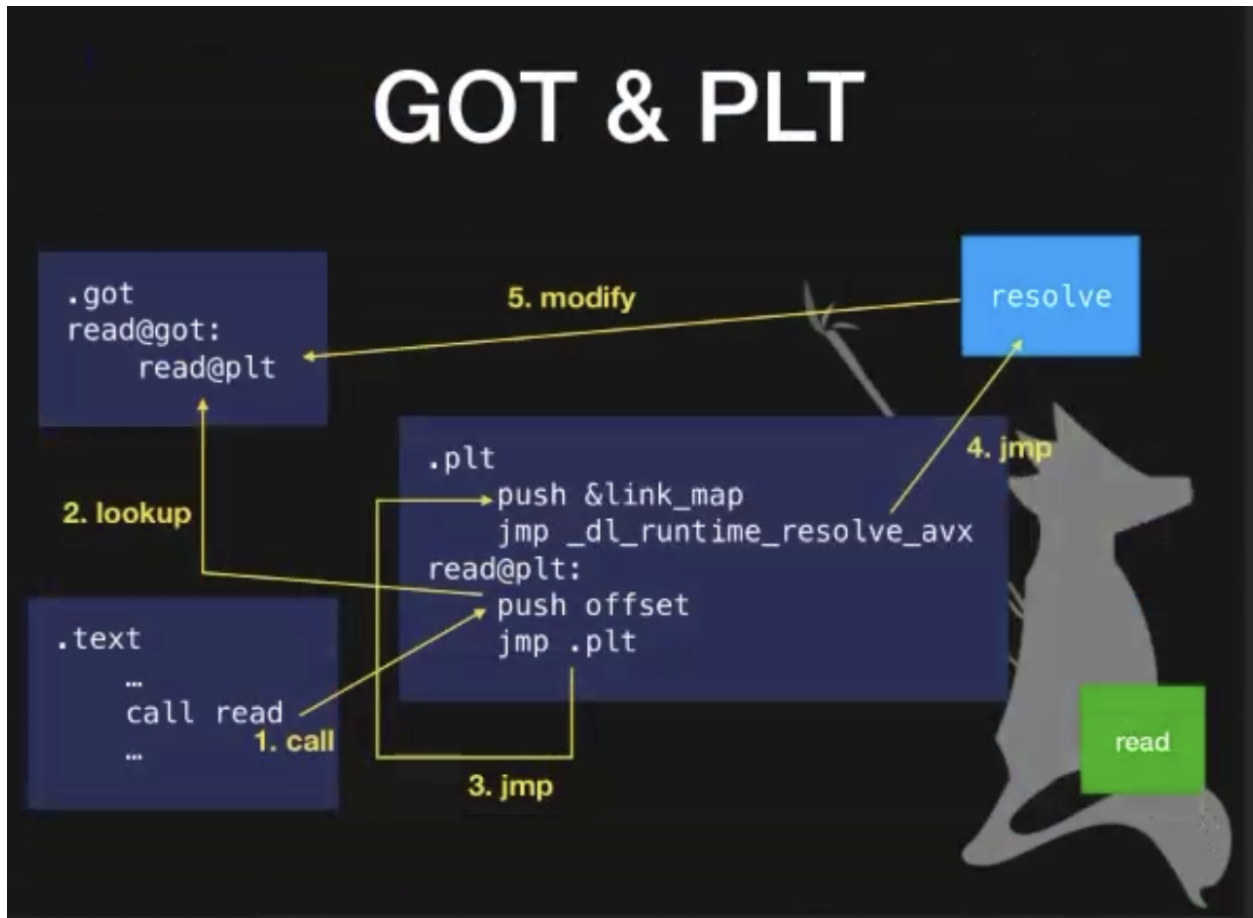
Execution

Lazy binding 機制

Lazy binding

Lazy binding – Rafael's Blog

- GOT, `_dl_fixup`
- link_map



Bookmark

- <https://www.slideshare.net/AngelBoy1>
- 🍷 <https://hackmd.io/@u1f383>
- 暗羽 Tags: Pwn <https://darkwing.moe/tags/Pwn/page/2/>

Linux Binary Exploitation

Angelboy @ AIS3 2017

<https://drive.google.com/file/d/1zdbCIhnMtxSlmrCQ7exCWVbmcpK75JJx/view>

<https://github.com/Naetw/CTF-pwn-tips>

第四章 技巧篇

https://firmianay.gitbooks.io/ctf-all-in-one/content/doc/4_tips.html

外星人 <https://hackmd.io/@n2bzaPikTJOQuazqdQUyWg/B1RCjvBZZ>

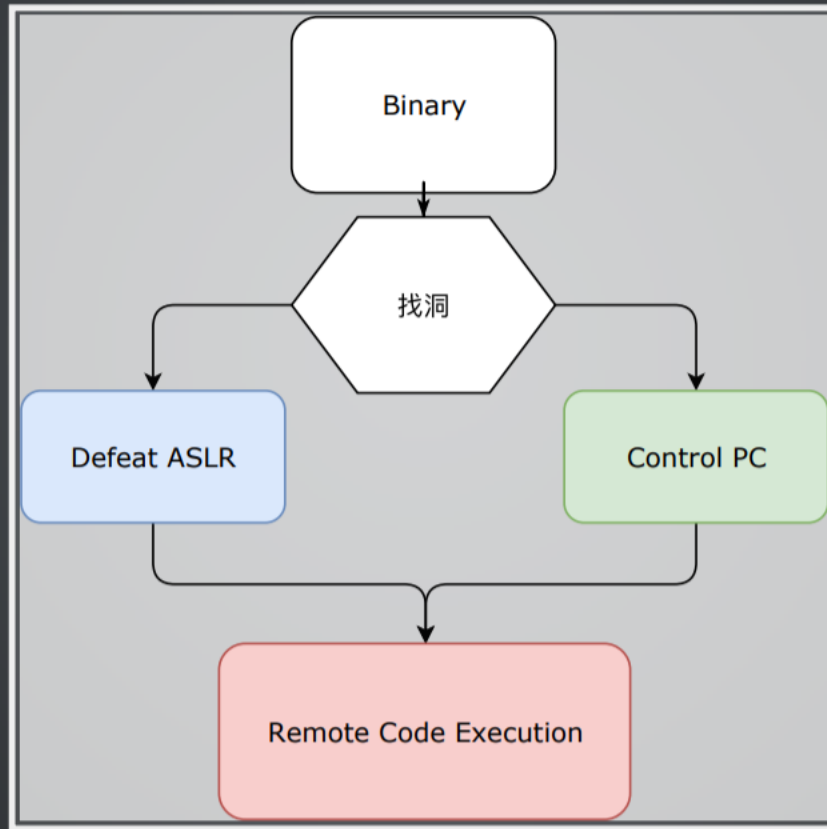
- https://www.google.com/search?q=https://hackmd.io/%40n2bzaPikTJOQuazqdQUyWg&sxsrf=ALeKk00JrQhEoxALSoouuzZ9Ny_jfHsKhQ:1607946170117&filter:
- pwn <https://hackmd.io/@n2bzaPikTJOQuazqdQUyWg/r1SNH07-M>

C++

<https://www.slideshare.net/AngelBoy1/pwning-in-c-basic>

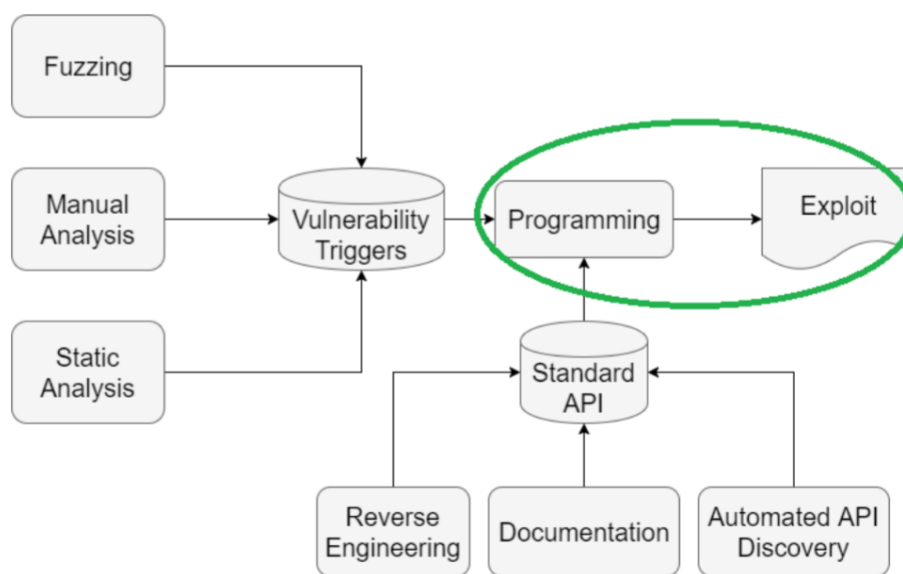
Exploit 方法論

Binary Exploitation in CTF



ref: R0D103 - david942j - 一發入魂One Gadget RCE

Exploit Development Process



@seanh / sean@vertex.re

ref: blackhat
Heap Layout Optimisation For Exploitation
Sean Heelan

漏洞分析 SOP

1. checksec
2. Seccomp
3. ghidra

根據利用的明確性決定優先順序 (如何剪枝)

前置知識

[https://mp.weixin.qq.com/s?](https://mp.weixin.qq.com/s?__biz=MzIxNjQ0MTEwMg==&mid=2247487838&idx=1&sn=1f75869457eaa19e8a43fb204728c77b&chksm=9788593fa0ffd02981b54)

[__biz=MzIxNjQ0MTEwMg==&mid=2247487838&idx=1&sn=1f75869457eaa19e8a43fb204728c77b&chksm=9788593fa0ffd02981b54](https://mp.weixin.qq.com/s?__biz=MzIxNjQ0MTEwMg==&mid=2247487838&idx=1&sn=1f75869457eaa19e8a43fb204728c77b&chksm=9788593fa0ffd02981b54)

各種處理器暫存器的順序,

為什麼需要 GOT 和 PLT ?

<https://stackoverflow.com/questions/43048932/why-does-the-plt-exist-in-addition-to-the-got-instead-of-just-using-the-got>

問題

1. 往漏洞研究領域專研，如 Fuzzing 與 Symbolic Execution 與自動化漏洞等領域，如果要往這些領域深入要如何下手？
2. 除了以上這些領域以外，還有什麼不可或缺技能呢？
3. 臺灣業界在做漏洞研究或相關領域有哪些公司？職涯上面可以有什麼發展？
4. Pwn 挖掘效率太慢怎麼？=> 每次了解漏洞之後製作一些掃描工具？
5. 如何分析 C++ 程式碼，看起來過於複雜是否有簡化的工具？

關於漏洞發生的地點

是否基礎 libc 剛出來的新功能都容易可以開採新漏洞？

看到的 addrT 可以用 ghidra 或 ida 下去分析

透過已知的漏洞的語意特性特徵？如 fmtstr, gets 等等常見漏洞等透過某種方式進行掃描？

分析現有 patch

`vimdiff <(xxd vulndb) <(xxd patch/067-round-090-team-5-ad-2-a772da960928cf64907fd225f60140be)`

`elfdiff vulndb patch/067-round-090-team-5-ad-2-a772da960928cf64907fd225f60140be`

分析工具

Ghidra Plugin 或 ida 有沒有快速掃描 buffer overflow 的工具？

uint unbr flow 掃描器？

https://keep.google.com/u/0/#NOTE/1dNGccJn2HzfOWs-mY2W0XLhWolAGvSXy_PQGH0Kyllw4KxczwHVAftTs3uk

透過 攻擊封包分析漏洞

cheat engine for linux

<https://github.com/korcankaraokcu/PINCE>

strace 可以 leak system call 等等的相關資料，如果程式讀取密碼會洩漏等等

- case HITCON CTF 2020 oshell

黑盒测试工具 Unicorn

ltrace
strace

ida text search 支援 regex 可以找到零進的，這個 gadgets 類似 pop 可以控制 sp 之後再跳到 ropchain

r2 有 webui 可以用

r2 找尋不安全的函數

cutter 可以找出不安全的函數，並且可以針對這裡優先分析，不過這裡顯示也缺少如 printf 也可能是不安全的，因此可能還需要一些人工介入或調整參數？

Address	Type	Library	Name	Safety
0x00000000	NOTYPE		_ITM_deregisterTMCloneTable	
0x00000000	NOTYPE		_ITM_registerTMCloneTable	
0x00000000	FUNC		__cxa_finalize	
0x00000000	NOTYPE		__gmon_start__	
0x00001330	FUNC		__isoc99_scanf	
0x00000000	FUNC		__libc_start_main	
0x00001220	FUNC		__read_chk	
0x00001250	FUNC		__stack_chk_fail	
0x00001200	FUNC		_exit	
0x00001320	FUNC		atoi	
0x00001300	FUNC		atol	
0x000011d0	FUNC		free	
0x000012b0	FUNC		gets	Unsafe
0x000012c0	FUNC		malloc	
0x000012f0	FUNC		memmove	
0x00001270	FUNC		memset	
0x00001310	FUNC		open	
0x00001260	FUNC		printf	
0x000011e0	FUNC		putchar	
0x00001230	FUNC		puts	
0x00001290	FUNC		read	
0x000012d0	FUNC		realloc	
0x000012e0	FUNC		setvbuf	
0x00001350	FUNC		sleep	
0x000012a0	FUNC		strcmp	
0x00001210	FUNC		strcpy	Unsafe
0x00001340	FUNC		strdup	
0x00001240	FUNC		strlen	Unsafe
0x00001280	FUNC		strncat	Unsafe
0x000011f0	FUNC		strncmp	

pwntool 使用

[17:35] sudo: 你可以試試看 p32(0x61616161)

[17:36] sudo: \x61 就是 b'a'

[17:36] sudo: 輸出之後就會被轉成 b'a'

[17:37] sudo: ROP 你比較需要擔心的是會不會被截斷那樣

o: ROP 你比較需要擔心的是會不會被截斷那樣

libc 的 environ 可以可以洩漏 stack 的地址

pwntools rop

可以用在堆積噴濺嗎？

pwnlib.memleak

- -----
-

__stack_chk_fail 漏洞利用

<https://nocbtlm.github.io/2020/04/28/stack-chk-fail相关利用/#exp>

<https://ctf-wiki.github.io/ctf-wiki/pwn/linux/mitigation/canary-zh/>

GOT and PLT for pwning.

<https://systemoverlord.com/2017/03/19/got-and-plt-for-pwning.html>

pwntools 斷點方法

- gdb script
- 在 script 裡面寫 input

z3solver

保護機制與繞過

Position Independent Executables (PIE)

Fortify Source

- 限制
 - fmt str 必須位於不可寫段
 - 使用 %10\$p 必須同時使用前面的參數

Canary

checksec 顯示 Canary enable 時，不代表所有函數都有保護，(是否有可能是手動修掉的?)

case

- BalsnXHitcon2020 vulndb error 函數

canary exploit

- one-by-one-crack canary
- TLS override
- __stack_chk_fail hijack
- fork leak
- canary leak
 - fmt str
 - off-by-one

- Stack Smashing Protect Leak
- 找尋 canary:
 - 經驗法則，最小位為 0x00 (little endian)
 - 如果為 multiple processes，因為子 process 共享同 canary，可以使子 process 洩漏 canary 來取得相關數值，child process crash 不會讓 parent crash

參考

- 栈溢出中关于canary的总结 <https://zhakul.top/archives/216>

Stack pivot

Stack Pivoting Tips

```

add     esp, 0xXXXX
ret
.....
sub     esp, 0xXXXX
ret
.....
ret 0xXXXX
.....
leave   ; (mov esp, ebp)
ret
.....
xchg    eXX, esp
ret
          
```

any gadgets that touch esp
will probably be of interest
for a pivot scenario

```

cmp     [ebp+arg_0], ebx
jnz     short loc_313066
mov     eax, [ebp+var_70]
cmp     eax, [ebp+var_84]
jb      short loc_313066
sub     eax, [ebp+var_84]
push    esi
push    eax
push    eax
push    edi
mov     [ebp+arg_0], eax
call    sub_31486A
test    eax, eax
jz      short loc_31306D
push    esi
lea     eax, [ebp+arg_0]
push    eax
mov     esi, 1D0h
push    esi
push    esi
call    sub_31411B
ret
loc_31306D:
; CODE XREF: sub_31306D+0
; sub_312FDB+56
push    0Dh
call    sub_31411B
loc_31306D:
; CODE XREF: sub_31306D+0
; sub_312FDB+49
call    sub_3140F3
test    eax, eax
jg      short loc_31307D
call    sub_3140F3
jmp     short loc_31308C
          
```

Stack pivot Gadget

<https://www.lazenca.net/plugins/servlet/mobile?contentId=23789591#content/view/23789591>

<https://www.mi1k7ea.com/2019/04/20/花式栈溢出之stack-pivoting/>

條件與方法

1. 條件 function 必須有 leave ; ret，轉移後的位置必須有 gadgets
2. esp 設定成預定轉移的位置
3. gadget leave ; ret

stack pivoting

1. 預 exploit 的 func 必須有 leave 指令來 pop rbp
2. rop chain 上面必須有目標 rbp 與一個 leave gadgets
3. 在第一次 leave 會把 rbp 給 pop 上去
4. 第二次 leave 會轉移 stack 到 rbp 上

洩漏 stack

- 透過 fmtstr 洩漏 ebp, env 等

ASLR vs PIE

PIE

PIE保护详解和常用bypass手段 - 安全客, 安全资讯平台

<https://www.anquanke.com/post/id/177520>

Pie繞過手法 pwn

- partial write
- leak base
- **vdso/vsyscall**: 主要作為 gadget 使用(地址固定)

pie only data and code but ASLR has stack/heap/library

ret2main

在只能控制 eip 不多的時候, 可以透過 ret2main 重複 pwn

RELRO (RELocation Read Only)

- **No RELRO**: 可寫 link_map 與 GOT
- **Partial RELRO**: 不可寫 link_map, 可寫 GOT
- **Full RELRO**: 不可寫 link_map, GOT
- ref <https://ithelp.ithome.com.tw/m/articles/10227381>

ROP 返回導向編程 (Shacham)

洩漏 libc

<https://book.hacktricks.xyz/exploiting/linux-exploiting-basic-esp/rop-leaking-libc-address>

通常先看 .got 是否可以利用，再來就是現成的 function，最後才考慮 ROP，
got 中若有 printf 可以利用 fmtstr，有 read 則有機會擴大 rop 的範圍

N byte 的 return address 部分複寫，或許可以透過 rop 工具快速找到可以用的部件

盲 pwn

- <https://ctf-wiki.github.io/ctf-wiki/pwn/linux/stackoverflow/medium-rop-zh/#blind-rop>

rop 除錯技巧，先找到一個可回顯的 gadget，比方說跳回 main 開頭可以重新執行等等，之後透過此 gadget 進行除錯。

可以透過隨機插入 (或者二分法) 到目前的 rop chain 的任意位置，如果沒有成功回顯代表程式出錯

該方法或許可以用於多執行緒進程的程式

通用的 gadget 可以 call 任意地址

ROP 工具使用

rp++

<https://github.com/0vercl0k/rp>

```
rp -f vulndb --rop=8
```

要求 syscall 有 ret 的syscall ; ret

./ROPgadget.py --binary=./binary.txt --multibr

pwntools 自動串串樂

注意事項:

- context.arch 請務必設定到正確的 arch 不然 calling convention 和 chain() 會爛掉

```
from pwn import *
c = constants
sh = ELF('/bin/sh')
libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')

sh.address += 0x400000
libc.address += 0x0000fdf1f8ff0000
context.arch = 'amd64'

rop = ROP([libc, sh])
rop.read(c.STDIN_FILENO, 0x123456, 0x22)
rop.write(c.STDOUT_FILENO, 0x123456, 0x22)
print(rop.dump())
payload = rop.chain() # put it into stack
payload
```

參見

<https://docs.pwntools.com/en/stable/rop/rop.html#rop-example>

linux 通用 gadgets

`__libc_csu_init()`

<https://wooyun.js.org/drops/一步一步学ROP之gadgets和2free篇.html>

https://github.com/firmianay/CTF-All-In-One/blob/master/doc/4.7_common_gadget.md

```
_init
_start
call_gmon_start
deregister_tm_clones
register_tm_clones
__do_global_dtors_aux
frame_dummy
__libc_csu_init
__libc_csu_fini
_fini
```

One Gadget

由 Dragon Sector 提出

[閒聊] OneGadgetTest(ogt) gdb plugin

<https://www.pttweb.cc/bbs/NetSecurity/M.1580371039.A.B89>

One Gadget 必須考慮 constraints 必須要成立才行

如果只能部份蓋寫

`one_gadget /lib/x86_64-linux-gnu/libc.so.6 --near exit,mkdir`

情況

- n gadgets exploit

- Case x86
 -
- Case x64
 - 2 gadgets: level 4 張元_Pwn-5 memo_manager

HITCON CTF 2017 Misc Two

X86 可以因為 x86 calling convention 而傳遞一個參數的函數，但是 x64 不行

除錯

- ld 或 libc 不合適的原因，啟動 sh 時會崩潰？
- gdb 斷點 syscall: catch syscall execve
- 在 libc 的 execve 下斷點檢查
- 為了方便除錯 透過 strace 來檢查是否有跑到 syscall
- strace -p pid
- 可以透過此方法驗證 syscall 是否正常運行

```
read(0, "\n", 15)
execve("/bin/sh", [0x10000
\307\350\371\227\1", "", "
70"], 0x7ffe0b44e780 /* 1
exit_group(127)
+++ exited with 127 +++
```

- 如果是透過盲猜 libc 的話，如工具 libc database search 透過 offset 找出 one gadget，為了確保正確請檢查 symbols offset 是否一致

格式化字串 fmtstr

<https://r888800009.github.io/software/security/binary/format-string-attack/>

問題

- https://ctf-wiki.github.io/ctf-wiki/pwn/linux/fmtstr/fmtstr_exploit-zh/#_9

格式化字串應用情景

1. 檢查 got 是否有 printf
2. 想辦法 leak printf address
3. 找到 system address
4. ret2system

例題

- babystack

特殊利用方法 `__malloc_hook()`

如果輸出極大可以觸發

例題

- ais3 club 2020 formatfree
- BalsnXhitcon 2020 交流賽 vulndb

參見

<https://www.jaybosamiya.com/blog/2017/04/06/adv-format-string/>

逆向工具

ghidra 有沒有自動 rename canary 的工具

也可以透過 ghidra 分析 stack frame ，再來就是動態除錯進行分析

一些逆向的技巧

ghidra 可以看 .plt ，可以知道那些函數被哪些地方使用，代表可能有相關漏洞等等

Ghidra 可以查看 activation record

ghidra 可查看 xref 列表: 對 func 右鍵可以看到 (ctrl + shift +f)

除錯

ghidra 似乎沒有把非可見字元跳脫，可能會影響分析，如 ANALYZER

如果洩漏某一個地址，並且為 stack 上的值，可以透過 ghidra 輔助分析如何取得 rbp

windbg

windbg 基本指令

<https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/getting-started-with-windbg>

windbg script??

x64dbg

輔助漏洞開發 exploit

<https://github.com/Andy53/ERC.Xdbg>

gdb

部份指令適用於 gdb peda 版本

show 10 個東西，分別對映兩種架構

x/10gx addr # x64

x/10wx addr # x32

顯示 string

x/s

暫存器

i r

找到特定數據

search 'c'

command

https://keep.google.com/u/0/#NOTE/1dNGccJn2HzfOWs-mY2W0XLhWolAGvSXY_PQGH0Kyllw4KxczwHVAftTs3uk

指令

<http://www.study-area.org/cyril/opentools/opentools/x1253.html>

插件

- GDB插件控制——切换pwndbg,peda,gef <https://www.jianshu.com/p/94a71af2022a>

AngelHeap

<https://github.com/scwuaptx/Pwngdb/tree/master/angelheap>

Pwngdb Heapinfo

<https://github.com/scwuaptx/Pwngdb>

問題

- 有無 gdb auto complete 插件?
- gdb 能否載入 header file 以便分析 heap?

/bin/sh

如果可以達成 system 的利用條件，其實不需要使用 `/bin/sh`，可以找到能夠符合 `sh` 或者 `;sh;` 之類的都是可以的

Library 當中也有 `/bin/sh` 的地址可以使用

如果使用 `one_gadgets` 或其他方法的話或許需要注意，sh 的版本
啟動 sh 時會崩潰，或許是 `ld` 或 `libc` 不合適的原因

除錯方法

- gdb 斷點 syscall: catch syscall execve
- 在 libc 的 execve 下斷點檢查

修改 lib 搜尋路徑

patchelf

另外也可以用 `pwndocker` 代替此問題

GOT 蓋寫技巧

反正就撞擊率，如 partial overwrite，如果只有 2 bytes，如 re-alloc pwnable.tw 透過碰撞的方式使 `atoi` 變成 `system`（在 `libc6_2.29-0ubuntu2_amd64` 時 兩者並不會相差太遠），這時只會受到 1/16 前面的 nibble 影響，進而碰撞 1/16 的機率。

RELRO

RELRO - A (not so well known) Memory Corruption Mitigation Technique

<https://mudongliang.github.io/2016/07/11/relro-a-not-so-well-known-memory-corruption-mitigation-technique.html>

Partial RELRO

`.init_array` `.fini_array` read only

緩解機制 Seccomp

`seccomp-tool` 可以查看

```

[ vagrant@racterubctf:~/ctf/balsnhitcon/release ] $ seccomp-tools dump ./wrapper
line  CODE  JT   JF      K
=====
0000: 0x20 0x00 0x00 0x00000004  A = arch
0001: 0x15 0x00 0x18 0xc000003e  if (A != ARCH_X86_64) goto 0026
0002: 0x20 0x00 0x00 0x00000000  A = sys_number
0003: 0x35 0x00 0x01 0x40000000  if (A < 0x40000000) goto 0005
0004: 0x15 0x00 0x15 0xffffffff  if (A != 0xffffffff) goto 0026
0005: 0x15 0x13 0x00 0x00000000  if (A == read) goto 0025
0006: 0x15 0x12 0x00 0x00000001  if (A == write) goto 0025
0007: 0x15 0x11 0x00 0x00000002  if (A == open) goto 0025
0008: 0x15 0x10 0x00 0x00000003  if (A == close) goto 0025
0009: 0x15 0x0f 0x00 0x00000005  if (A == fstat) goto 0025
0010: 0x15 0x0e 0x00 0x00000009  if (A == mmap) goto 0025
0011: 0x15 0x0d 0x00 0x0000000a  if (A == mprotect) goto 0025
0012: 0x15 0x0c 0x00 0x0000000b  if (A == munmap) goto 0025
0013: 0x15 0x0b 0x00 0x0000000c  if (A == brk) goto 0025
0014: 0x15 0x0a 0x00 0x0000000e  if (A == rt_sigprocmask) goto 0025
0015: 0x15 0x09 0x00 0x00000011  if (A == pread64) goto 0025
0016: 0x15 0x08 0x00 0x00000014  if (A == writev) goto 0025
0017: 0x15 0x07 0x00 0x00000015  if (A == access) goto 0025
0018: 0x15 0x06 0x00 0x0000003c  if (A == exit) goto 0025
0019: 0x15 0x05 0x00 0x0000009e  if (A == arch_prctl) goto 0025
0020: 0x15 0x04 0x00 0x000000e6  if (A == clock_nanosleep) goto 0025
0021: 0x15 0x03 0x00 0x000000e7  if (A == exit_group) goto 0025
0022: 0x15 0x02 0x00 0x00000101  if (A == openat) goto 0025
0023: 0x15 0x01 0x00 0x0000012e  if (A == prlimit64) goto 0025
0024: 0x15 0x00 0x01 0x00000142  if (A != execveat) goto 0026
0025: 0x06 0x00 0x00 0x7fff0000  return ALLOW
0026: 0x06 0x00 0x00 0x00000000  return KILL

```

ELF 修改保護

- NX bit

如何啟用 nx?

透過 patchkit 處理 ELF

mmap 透過 mprotect 改成 rwx 以便利用

ELF

NX(No-eXecute)的实现分析

<https://hardenedlinux.github.io/system-security/2016/06/01/NX-analysis.html>

```
731  /* Legal values for p_flags (segment flags).  */
732
733  #define PF_X          (1 << 0)      /* Segment is executable */
734  #define PF_W          (1 << 1)      /* Segment is writable */
735  #define PF_R          (1 << 2)      /* Segment is readable */
736  #define PF_MASKOS     0x0ff00000    /* OS-specific */
737  #define PF_MASKPROC   0xf0000000    /* Processor-specific */
738
```

ref: <https://github.com/lattera/glibc/blob/895ef79e04a953cac1493863bcae29ad85657ee1/elf/elf.h#L733-L737>

檢視 elf 差異可以透過工具 elfdiff

exit

<https://n132.github.io/2019/05/15/2019-05-15-EXIT-HCTF2018-the-end/#tls-dtor-list>

pwndocker

<https://github.com/skysider/pwndocker>

可以方便不用切換 ubuntu 的版本嘛？

Scanf trick

- scanf 吃 null byte
 - scanf termination character
 - 規格寫直到 whtiespace (\0 是 whitespace 嗎？)
- %u input without modity, input '-' or '+'
 - ref: <https://www.freebuf.com/articles/others-articles/134271.html>

gdb peda

vmmap 可以代替 info proc mapping，並且可以看到權限能不能寫入，比方說 rwxp

lldb 類似 vmmap

image dump sections

如何有效提昇 pwn 的速度？？？

heap spray ?

洩漏遠程的 lib

DynELF

越界寫入漏洞

- death_note

sudo: 話說我在 arch 上面 pwn 都會遇到坑

[01:15] sudo: 比方說 ubuntu 上面的 heap 可以執行

```
Start      End      Perm      Name
0x08048000 0x08049000 r-xp      /home/ubuntu/death_note
0x08049000 0x0804a000 r-xp      /home/ubuntu/death_note
0x0804a000 0x0804b000 rwxp      /home/ubuntu/death_note
0x0804b000 0x0806d000 rwxp      [heap]
0xf7dec000 0xf7fc1000 r-xp      /lib/i386-linux-gnu/libc-2.27.so
0xf7fc1000 0xf7fc2000 ---p      /lib/i386-linux-gnu/libc-2.27.so
0xf7fc2000 0xf7fc4000 r-xp      /lib/i386-linux-gnu/libc-2.27.so
0xf7fc4000 0xf7fc5000 rwxp      /lib/i386-linux-gnu/libc-2.27.so
0xf7fc5000 0xf7fc8000 rwxp      mapped
0xf7fc8000 0xf7fd1000 rwxp      mapped
0xf7fd1000 0xf7fd4000 r--p      [vvar]
0xf7fd4000 0xf7fd6000 r-xp      [vdso]
0xf7fd6000 0xf7ffc000 r-xp      /lib/i386-linux-gnu/ld-2.27.so
0xf7ffc000 0xf7ffd000 r-xp      /lib/i386-linux-gnu/ld-2.27.so
0xf7ffd000 0xf7ffe000 rwxp      /lib/i386-linux-gnu/ld-2.27.so
0xf7ffe000 0xffffd000 rwxp      [stack]
gdb-peda$
```

[01:17] sudo: 然後 arch 的 heap 不知道為什麼就是不行

```
gdb-peda$ vmmap
Start      End      Perm      Name
0x08048000 0x08049000 r-xp      /home/arch/git/CTF-Solve/pwnable.tw/deat
death_note
0x08049000 0x0804a000 r--p      /home/arch/git/CTF-Solve/pwnable.tw/deat
death_note
0x0804a000 0x0804b000 rw-p      /home/arch/git/CTF-Solve/pwnable.tw/deat
death_note
0x0804b000 0x0806d000 rw-p      [heap]
0xf799d000 0xf7db0000 r--p      /usr/lib32/libc-2.32.so
0xf7db0000 0xf7f17000 r-xp      /usr/lib32/libc-2.32.so
0xf7f17000 0xf7f89000 r--p      /usr/lib32/libc-2.32.so
0xf7f89000 0xf7f8b000 r--p      /usr/lib32/libc-2.32.so
0xf7f8b000 0xf7f8d000 rw-p      /usr/lib32/libc-2.32.so
0xf7f8d000 0xf7f91000 rw-p      mapped
0xf7f91000 0xf7fce000 r--p      [vvar]
0xf7fce000 0xf7fd0000 r-xp      [vdso]
0xf7fd0000 0xf7fd1000 r--p      /usr/lib32/ld-2.32.so
0xf7fd1000 0xf7ff0000 r-xp      /usr/lib32/ld-2.32.so
0xf7ff0000 0xf7ffb000 r--p      /usr/lib32/ld-2.32.so
0xf7ffb000 0xf7ffd000 r--p      /usr/lib32/ld-2.32.so
0xf7ffd000 0xf7ffe000 rw-p      /usr/lib32/ld-2.32.so
0xffffd000 0xffffe000 rwxp      [stack]
gdb-peda$
```

[01:17] sudo: 就算我換了 glibc 也是一樣

[01:18] sudo: 想說 kernel 是不是有相關參數預設是開啟的

一些關於 pwntool

如果需要遠端存取，可能需要使用 tmux

```
context.terminal = ['tmux', 'splitw', '-h']
context.terminal = ['tmux', 'splitw', '-v']
```

ref: <http://brieflyx.me/2015/python-module/pwntools-advanced/>

Heap exploit

分析方法

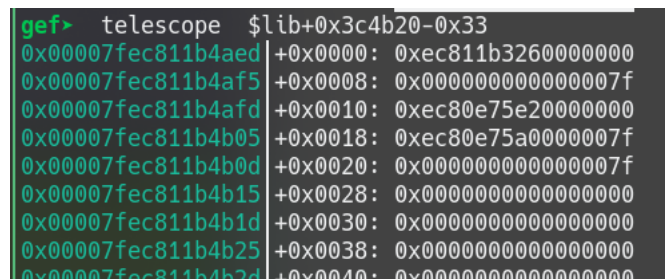
- 透過 gdb GEF 的 heap 功能分析 heap
 - heap chunks
 - heap-analysis-helper 可以追蹤 malloc 和 free
- gdb-peda\$ heapinfo
- <https://github.com/danigargu/heap-viewer>

libc 洩漏

- unsorted bin: **main_arena**

攻擊方式

- 嘗試 overlap chunk 達成控制 chunk，如程式有 UAF 漏洞 可達成
 - 相關條件: null byte off by one
 - <https://www.anquanke.com/post/id/208407#h2-12>
- unlink 任意地址複寫？
- hijack hook
 - 重點: 在 **malloc_hook** 前面找到極小值作為 fake chunk
 - chunk 大小通常為 0x7f
 - 該方法是透過偏移記憶體上面的值來獲得可以接受的大小 malloc_hook 的位置減去非 0x8 的大小，如 -0x3
 - 0x00007f1234564c38 左移 3 bytes ⇒ 0x7f, 0x1234564c38 可以得到 0xf7 使用
 - main_arena: \$lib+0x3c4b20-0x33



```
gef> telescope $lib+0x3c4b20-0x33
0x00007fec811b4aed +0x0000: 0xec811b3260000000
0x00007fec811b4af5 +0x0008: 0x000000000000007f
0x00007fec811b4afd +0x0010: 0xec80e75e20000000
0x00007fec811b4b05 +0x0018: 0xec80e75a0000007f
0x00007fec811b4b0d +0x0020: 0x000000000000007f
0x00007fec811b4b15 +0x0028: 0x0000000000000000
0x00007fec811b4b1d +0x0030: 0x0000000000000000
0x00007fec811b4b25 +0x0038: 0x0000000000000000
0x00007fec811b4b2d +0x0040: 0x0000000000000000
```

- 其他想法: 如果 offset 用 +1 是否也可以呢？但是穩定性不夠，是因為最低 byte 不保證低於 0x80
- 如果要寫入 程式的 text，可能在 0x400000，也可以嘗試移動到 0x40, 0x0000 來達成一樣的效果任意 malloc
- objdump -T ./libc-2.23.so | grep 'hook'

- <https://github.com/Naetw/CTF-pwn-tips#hijack-hook-function>
- 各種 hook 的參數可以參見 malloc.h
- <https://android.googlesource.com/platform/bionic/+/master/libc/include/malloc.h>
- __malloc_hook 可以蓋寫，當中如 printf 過大 %50000c 左右可以觸發 __malloc_hook，當中需要找機會呼叫 malloc？
- ref: angelboy

Heap overflow

- Some useful global variable
 - __free_hook、__malloc_hook、__realloc_hook
 - free,malloc,realloc 前會先檢查是否上面三個全域變數是否有值，若有則當 func ptr 執行
 - __malloc_initialize_hook
 - ptmalloc_init 時（第一次執行 malloc 會用），會檢查該變數是否有值，若有則當 func ptr 執行
 - __after_morecore_hook
 - 在跟用 brk 跟系統要 heap 空間時，會檢查該變數是否有值，若有則當 func ptr 執行

- hook 不起作用，可能是因為執行的時候程式崩潰了，或許可以追到 system call 的位置檢查是回傳值
- gdb: catch syscall execve

相關議 [HITCON 2020] Last orders at the House of Force

https://www.youtube.com/watch?v=SG12uOOauNw&list=PLJXs_C_JomgRlVSTWgWcdR8srtQrbD2Hf&index=5&t=837s

heap 任意地址寫入

fastbin attack

<http://reborn2266.blogspot.com/2011/12/glibcmalloc-hook.html>

Day17: [Pwn] Educational Heap Exploitation - part 2 ddaa

<https://ithelp.ithome.com.tw/articles/10223900>

如果可以執行 heap 段，原則上只要控制 eax 與 ecx 指向可寫的地址，以及最後 chunk 的結尾跳過，基本上可以把 heap 裡面的值當成 NOP slide。

例題: alive_note pwnable.tw

關於 Heap Protector 文檔指出由 glibc 進行保護，其他如 stack 文檔指出由 stack 保護(參見 [ubuntu 文檔](#))

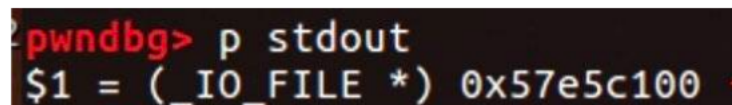
一些 heap 常見的招數 通常會用到 malloc 回傳任意地址

<https://heap-exploitation.dhavalKapil.com/attacks/>

為了輸出，可能需要蓋寫 stdout

<https://paper.seebug.org/935/>

可以撞 1/16 的機率



```
pwndbg> p stdout  
$1 = (_IO_FILE *) 0x57e5c100
```

其他關於 heap 的手法可以參見 ctf wiki

https://ctf-wiki.github.io/ctf-wiki/pwn/linux/glibc-heap/house_of_einherjar-zh/

Glibc Heap Related	▼
Introduction to Heap	
Heap Overview	
Heap Related Data Structure	
In-depth Understanding of Ptmalloc2	>
Heap Overflow	这.
Heap-based Off-By-One	
Chunk Extend/Overlapping	
Unlink	
Use After Free	
Fastbin Attack	
Unsorted Bin Attack	
Large Bin Attack	
Tcache Attack	
House of Einherjar	
House of Force	
House of Lore	
House of Orange	
House of Rabbit	
House of Roman	

free 之後 realloc 似乎可以改變 chunk的資訊？

- pwnable.tw re-alloc

UAF

UAF 利用方法

利用方法實做部份

如何偽造 fake chunk

- gdb 如何定位 heap 區段
 - 在 bss 段紀錄 addr 如 raas hackme.innndy.tw，可以直接用 gdb 直接檢查 bss 段上的資料
 - 在 stack 紀錄 addr，通常配置到 local variable 等等 stack 上面的數值，可以透過 gdb 定位此

- 在 heap 段紀錄 addr
- 如何對齊 fake chunk，應用 fastbin 特性等等

forging chunk 構造假的 free chunk

- pwn ctf
 - pwno forging_chunk

構造方法參見

https://heap-exploitation.dhaval kapil.com/attacks/forging_chunks

- 技巧
 - 可以透過 gef> heap chunk 來得知 chunk 的大小，與 chunk 的位置與內容

Unlink

- 前置概念
 - Top chunk: 尚未分配出去的另一大塊 chunk
- 利用 bypass check 的一些方法
 - 找到特定版本的 malloc.c 並且分析裡面每個錯誤？
 - 比方說想找 glibc 2.23 可以看
 - <https://github.com/bminor/glibc/blob/glibc-2.23/malloc/malloc.c>
 - gdb 是否可以套用 header 以便分析結構？？
- 觸發條件
 - 的 fd 與 bk 的值必須指向可寫區段
 - 合併之後會進行 unlink 移除 bin 重複的 chunk

Mechanism of glibc malloc

- Merge freed chunk
 - 合併流程
 - 如果上一塊事 freed
 - 合併上一塊 chunk，並對上一塊做 unlink
 - 如果下一塊是
 - top : 合併到 top
 - 一般 chunk :
 - freed : 合併下一塊 chunk，並最下一塊做 unlink，最後加入 unsortedbin
 - inuse : 加入 unsortedbin

- ref Angel Boy Heap exploitation p30
- 目的
 - 達成任意寫入
- 限制
 - fastbin 無法觸發
 - 可能需要 unsortbin 的位置 ???
 - 構造 unlink 時 fd, bk 需要指向 chunk → pre 的位置 需要另外構造 chunk
- 相關
 - House Of Einherjar **向後合併**
- 習題 pwno unlink

ref:

<https://wooyun.js.org/drops/堆溢出的unlink利用方法.html>

注意事項:

- overflow 進行 unlink 的 chunk 是否不能為 top chunk?

glibc Heap

dldmalloc

分析程式流程

https://b0ldfrev.gitbook.io/note/pwn/_int_malloc

- malloc 報錯透過 malloc_printerr
 - 或許可以在 gdb 在這裡下斷點以方便追蹤堆疊
 - 方便分析鄰近的程式碼

程式流程

- malloc
-
- free

概念

- victim: bin 的最後 chunk

merge 時, 如果有辦法偽造負值的大小可以往後 marge

HEAP 詳盡整理 (Poming huang)

<https://hackmd.io/@5Mo2wp7RQdCOYcqKeHl2mw/B1AKOwKSz>

malloc_trim() 的 main_arena

```

21 |
22 | if (DAT_004c4144 < 0) {
23 |     FUN_00185460();
24 | }
25 | local_44 = 0;
26 | local_50 = &DAT_004c4b20;
27 | do {
28 |     if (DAT_004c9740 == 0) {
29 |         bVar10 = *local_50 == 0;
30 |         *local_50 = *local_50 ^ (uint)bVar10 * (*local_50 ^ 1);
31 |     }

```

```

4551 |
4552 | int
4553 | __malloc_trim (size_t s)
4554 | {
4555 |     int result = 0;
4556 |
4557 |     if (__malloc_initialized < 0)
4558 |         ptmalloc_init ();
4559 |
4560 |     mstate ar_ptr = &main_arena;
4561 |     do
4562 |     {
4563 |         (void) mutex_lock (&ar_ptr->mutex);
4564 |         result |= mtrim (ar_ptr, s);
4565 |         (void) mutex_unlock (&ar_ptr->mutex);
4566 |
4567 |         ar_ptr = ar_ptr->next;
4568 |     }
4569 |     while (ar_ptr != &main_arena);
4570 |
4571 |     return result;
4572 | }
4573 |
4574 |

```

malloc(size) , heap 實際大小 = size + 0x10

最頂端有一個 top chunk , top chunk 為 malloc 之後另外一些未被分配的空間, 並且保存 top chunk 的 size, 相關利用方法包含 House of Force , 蓋過 top chunk 的 size 成大值

關於 heap overflow , fake chunk 可以採用 house of spirit

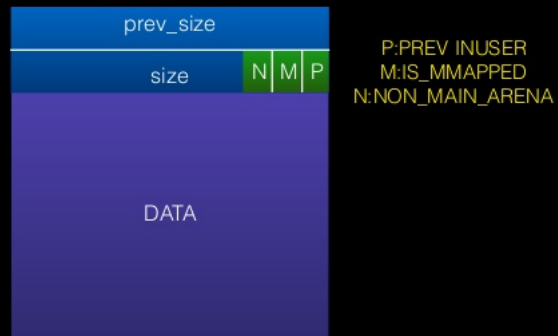
相關例題 Spirited Away pwnable.tw

必須注意三個在 size 的 flag

- 該 chunk 的大小, 其中有三個 flag
 - PREV_INUSE (bit 0) : 上一塊 chunk 是否不是 freed
 - IS_MMAPPED (bit 1) : 該 chunk 是不是由 mmap 所分配的
 - NON_MAIN_ARENA (bit 2) : 是否不屬於 main arena

Mechanism of glibc malloc

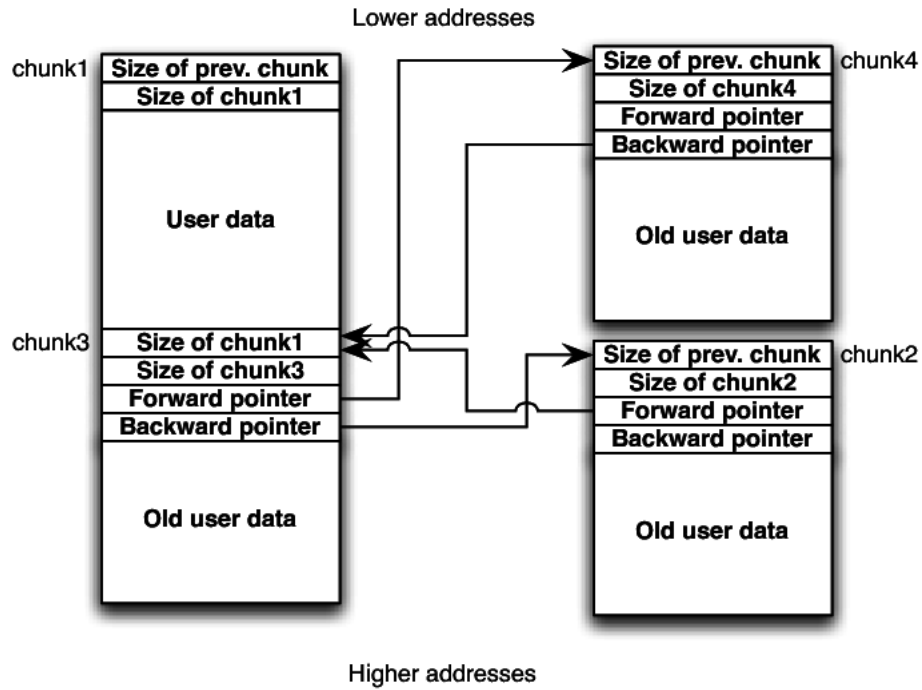
- Allocated chunk



ref: Angel Boy

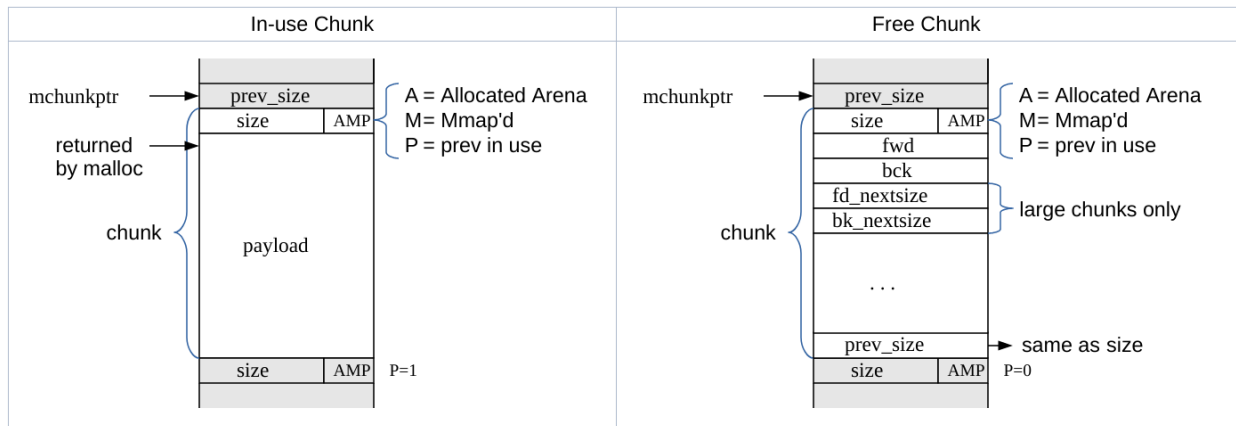
關於 heap 的結構，可以看到被使用的 chunk 含有三個 part 前面的

- 使用中的
 - prev chunk size
 - cur chunk size
 - data
- 未使用的，部份 data 被取代為 fd 和 bk
 - prev chunk size
 - cur chunk size
 - **fd pt**
 - **bk pt**
 - old data

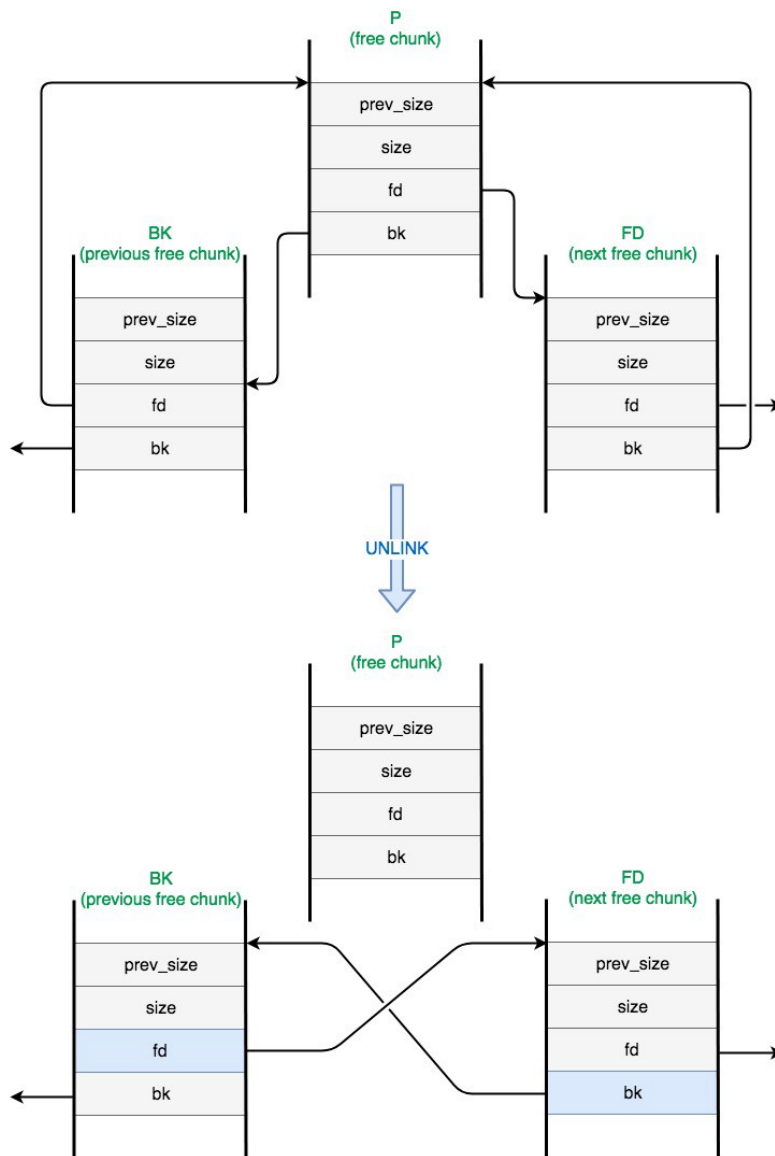


Heap containing used and free chunks

figure ref: Younan, Yves & Joosen, Wouter & Piessens, Frank. (2006). Efficient Protection Against Heap-Based Buffer Overflows Without Resorting to Magic. 4307. 379-398. 10.1007/11935308_27.



ref: <https://sourceware.org/glibc/wiki/MallocInternals>



<https://medium.com/@airman604/protostar-heap-3-walkthrough-56d9334bcd13>

類似概念

- pwnable.tw applestore

tcache

基本知識

- tcache 位於 heap 處並且有 64 個 linkedlist

相關手法

- CVE-2017-17426 輸入一個接近 SIZE_MAX 回傳 tcache bin，適用於修補時間: Date: Thu Nov 30 13:31:45 2017 +0100 以前的 libc 2.26
 - 該手法如果傳入一個負值，似乎可以解讀成 0xffffffffffffff

- 例題

- 神頓 2020 babyheap (該解題非公開)

<https://hackmd.io/YXqyCoRGtzOP0PoKP9fKHA?both#babyheap>

- 洩漏 libc 地址

- 可作 malloc hook hijack
- 通常 stdout 都有使用，而 malloc 時候原始的值會被當成下一個 chunk，也就是說如果 malloc 之後後面是 0 就不會無效地址問題，

參見

- https://ctf-wiki.github.io/ctf-wiki/pwn/linux/glibc-heap/tcache_attack-zh/

Tcache Exploitation

<https://www.slideshare.net/AngelBoy1/tcache-exploitation-127268389>

TCACHE exploitation. 最近越來越排斥在medium上寫很技術性的東西，因為我發現自己點開medium也... | by bering | Medium
<https://medium.com/@ktecv2000/tcache-exploitation-871044f8b210>

https://blog.csdn.net/qg_38154820/article/details/106330125

利用方向

- 泄露libc。这个肯定是要想办法用unsortedbin来实现，比较好的思路是合并堆，释放到unsortedbin，且能使unsortedbin和未释放的堆重叠。由于存在null of byte。可以向前overlapping。注意将被更改size的请求大小一定是0x?f0。

HAL heap

參考文獻

相關利用手法

- My journey on SMBGhost angelboy
- https://hitcon.org/2020/slides/My_journey_on_SMBGhost.pdf

Shellcode 編碼

參考 pwntools 文檔

<https://docs.pwntools.com/en/stable/encoders.html>

pwnlib.encoders.encoder.encode

pwnlib.encoders.i386.xor.i386XorEncoder

<https://docs.pwntools.com/en/stable/encoders.html#pwnlib.encoders.i386.xor.i386XorEncoder>

Shellcode 轉可見字元

<https://github.com/VincentDary/PolyAsciiShellGen>

```
msfvenom -a x86 --platform linux -p linux/x86/exec CMD=/bin/sh -e x86/alpha_mixed BufferRegister=ECX -f raw -o /dev/stdout |  
python -c 'from pwn import *; print(disasm(input().encode()))'
```

X64 可能不能使用以上工具，可以採用以下工具，但是容易過長

https://github.com/rcx/shellcode_encoder

再來是一些產生 printable 字元請參考文章

BufferRegister 為指向 shellcode 的 register

<https://www.anquanke.com/post/id/85871>

理論上可以透過 xor 來寫入記憶體，可以寫一小段程式碼寫入特定的直到記憶體，參見 [alive_note.py](#) 的寫入器 (注: 其實 alive_note 可以直接透過 sys_read 而不是寫入器，利用上可能會比較快)

可以查表 https://nets.ec/Ascii_shellcode

<http://phrack.org/issues/57/15.html#article>

pwnlib.encoders.encoder.alphanumeric

pwnlib.encoders.encoder.printable

arm

- thumb gadgets (arm32)

參考

<https://wooyun.js.org/drops/一步一步学ROP之Android ARM 32位篇.html>

android 的 ASLR 為偽 ASLR 透過 zygote fork 處理

aarch64

arm64

TTBR 濫用來提升權限

Linux

修改記憶體權限權限

- 可以透過 mprotect 可以修改 mmap 的權限

linux kernel

- init_module

https://danielmaker.github.io/blog/linux/start_kernel.html

逆向工具

- extract-vmlinux
- wget <https://raw.githubusercontent.com/torvalds/linux/master/scripts/extract-vmlinux>
- `chmod +x extracted-kernel`
- `./extract-vmlinux /boot/vmlinuz-linux > extracted-kernel`

How to extract and disassemble a Linux kernel image (vmlinuz)

<https://blog.packagecloud.io/eng/2016/03/08/how-to-extract-and-disassemble-a-linux-kernel-image-vmlinuz/>

參考

<https://xz.aliyun.com/t/7625>

IOS kernel

tools

- memctl

windows kernel

struct

- <https://www.vergiliusproject.com/>

<https://github.com/hacksystem/HackSysExtremeVulnerableDriver>

PTE (page table entry) 濫用

Windows PEB

Process Environment Block

Thread

TLS

<https://github.com/torvalds/linux/blob/master/include/net/tls.h>

https://hackmd.io/@oscarshiang/linux_ktls

如何對 multi thread 或 process 的程式進行 exploit?

跟隨子 process

set follow-fork-mode child

fork 的 canary 相同

fs:0x28

fork 的 process layout 與 parent 相同

child process 崩潰，不影響 parent

File struct

p *(struct _IO_FILE_plus *) stdout

vtable

_IO_file_jumps

似乎唯獨？

- __start__libc_IO_vtables
- __stop__libc_IO_vtables

FSOP

- glibc/libio/genops.c

2.24

- libio/libioP.h IO_validate_vtable

_IO_vtable_check (void)

<https://elixir.bootlin.com/glibc/latest/source/libio/vtables.c>

2.28

- __libc_IO_vtables 無法利用

CET

shadow stack

找出 shadow stack 的 return address

(gdb) monitor help sde

```
k-get-regs          - Dump the mask registers
k-get-reg N         - Dump the mask register N
```

<https://software.intel.com/content/www/us/en/develop/articles/debugging-applications-with-intel-sde.html>

例題

- ais3 EOF 2021 final RevengeOfTheIntel

對齊

- 部分版本 libc 需要對齊 16 bytes (movaps) ，如果出錯的話，嘗試多 ret, pop, sub 等等方法來控制 rsp

GoLang

golang 預設為 static linking

ref:

https://eqqie.cn/index.php/laji_note/1418/