一發入魂

One Gadget RCE

david942j @ HITCON CMT 2017

Before Start

Q: 如何製作好的簡報？

三個訣竅

1. 有貓
2. 很多貓
3. 滿滿的貓

Who Am I

- david942j
- CTF 電競選手
  - 哪兒有獎金就往哪打

Introduction

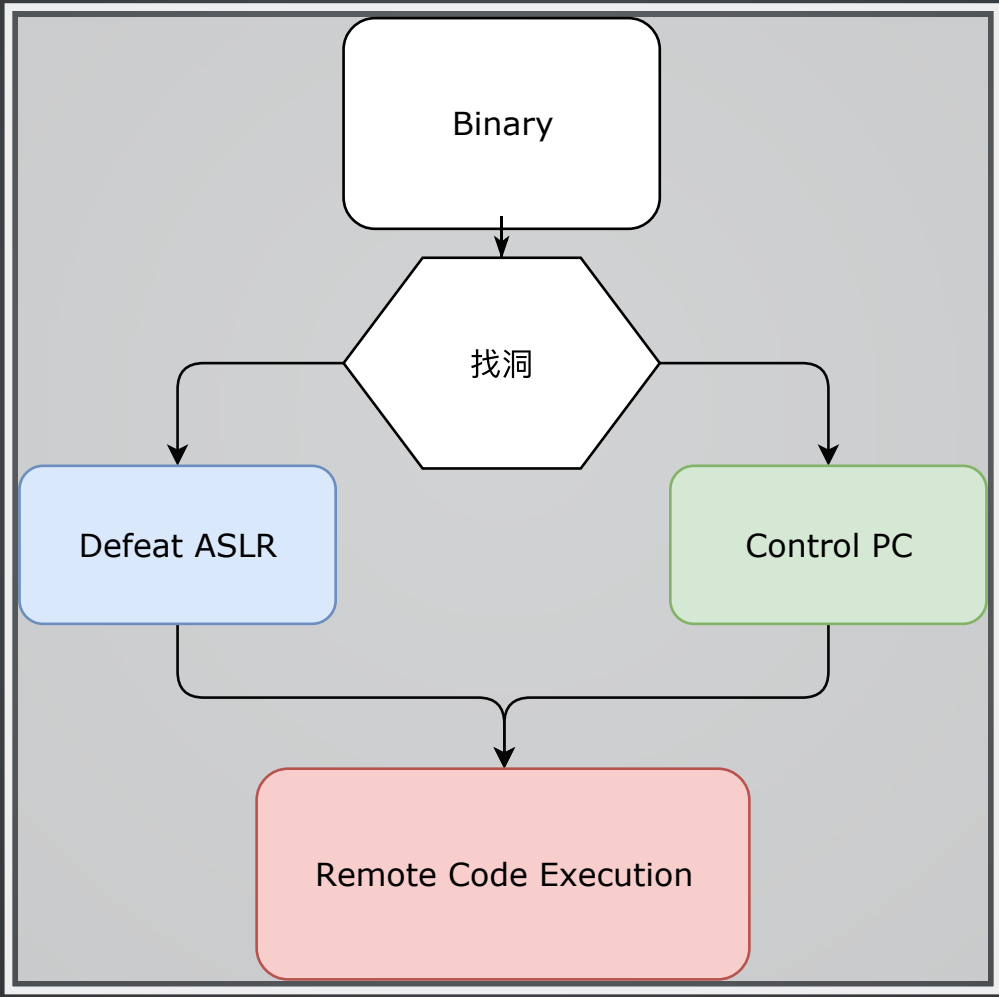
Hacker の日常

吃飯 睡覺 打東東

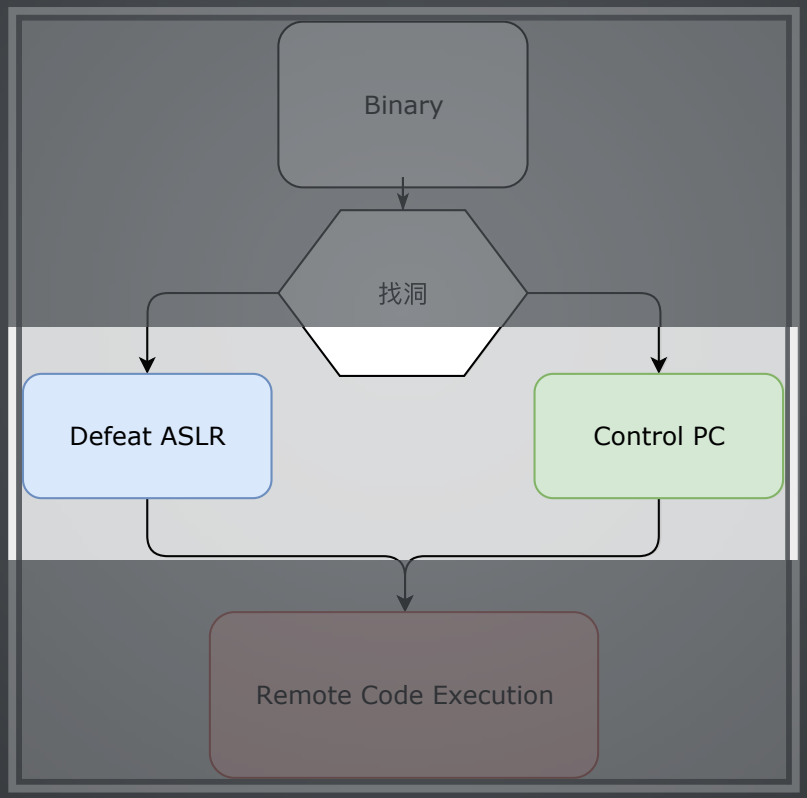
東東

- Binary Services (PE / ELF)
  - PE - Windows
  - ELF - Linux
- Web Browser
- OS Kernel (privilege escalation)
- VM (VM escape)
- CTF

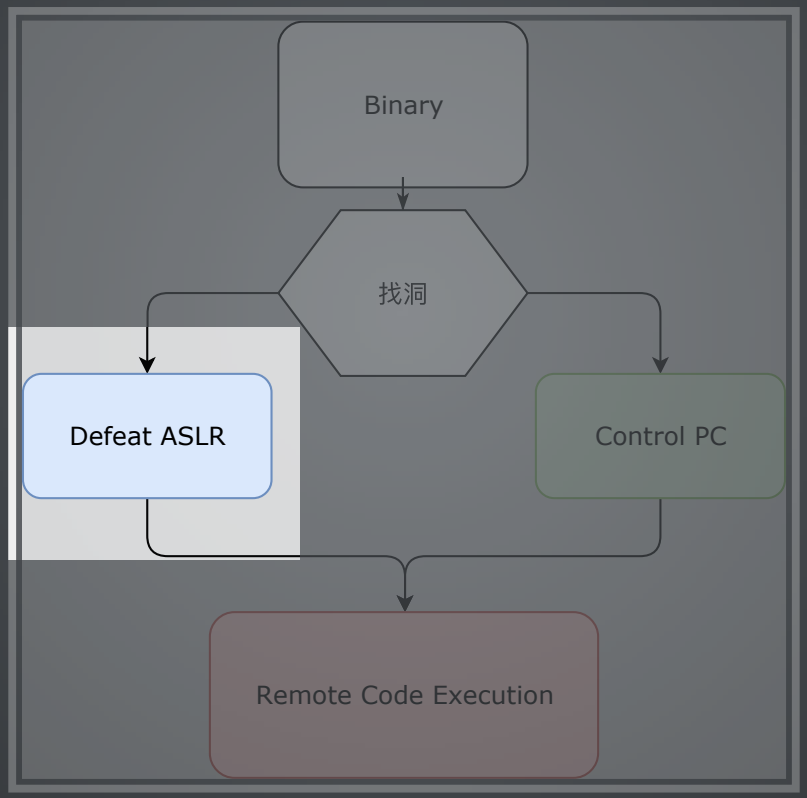
Binary Exploitation in CTF



漏洞利用目標有兩種



Defeat ASLR



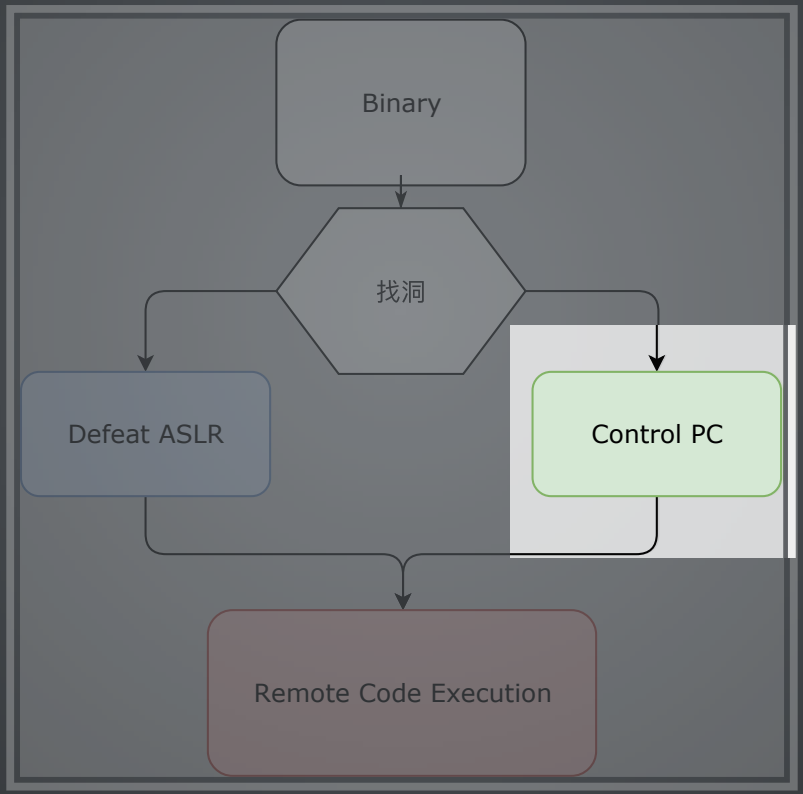
ASLR

- 隨機化配置記憶體位址
- Linux kernel 2.6.12 (2005, June)

Defeat ASLR

- 利用漏洞洩露記憶體位址
- Information leak

Control PC



Control PC

~~控制電腦~~

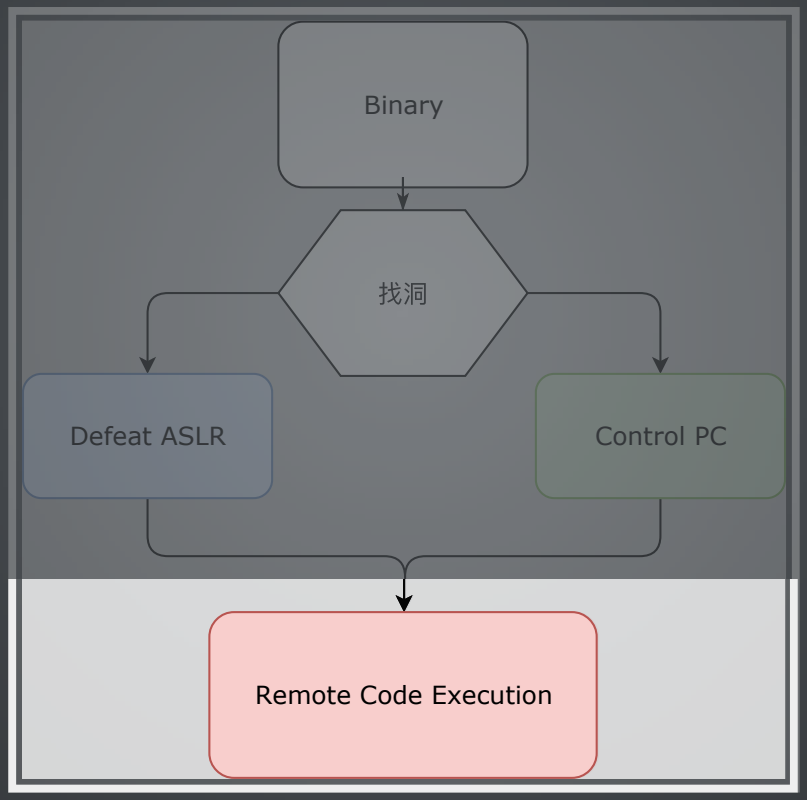
Program Counter

控制執行哪一行組合語言

Control PC = 控制程式流程

任意跳

Remote Code Execution



DeASLR + Control PC = RCE

余憶童稚時

RCE = `system("sh")`

呼叫 `system("sh")`

- DeASLR: 洩漏 `system` 函式的位址
- 任意跳: 跳到 `system`
- 同時控制參數為 "sh"
  - 可能辦不到

One Gadget

功能

跳上去就 RCE

不需控參數！

使用時機

(1)

已知函式庫的記憶體位址

DeASLR

(2)

PC 可控

(3)

已知肉雞的函式庫版本

才能準確知道要跳哪

<u>可以用猜的</u>

原理

函式庫裡有些函式會呼叫

```
execve("/bin/sh", argv, envp)
```

談談 execve 這玩意兒

execve

```
int execve(const char *file, char *const argv[], char *const envp[])
```

- 執行檔案的 syscall
- `main(int argc, char **argv, char **envp){}`
- `execve(file, argv, envp)`

Example

執行 `ls -lh`

```
#include <unistd.h>
int main() {
  char *const argv[] = {"ls", "-lh", NULL};
  char *const envp[] = {NULL};
  execve("/bin/ls", argv, envp);
}
```

Demo

argv 與 envp

- 根據文件是個 NULL 結尾的 array of pointers
  - `char *s[] = {ptr1, ptr2, ..., NULL}`
  - `execve(file, {NULL}, {NULL})`
- 小彩蛋(?) - 直接 NULL 也可以唷
  - `execve(file, NULL, NULL)`

聊聊 argv

argv

- 執行參數
- 根據寫法不同，`argv=NULL` 可能會死
- Demo
- 好消息: /bin/sh 不會死
  - ✓ `execve("/bin/sh", NULL, NULL)`

聊聊 envp

envp

- 環境變數
- `environ` 可以取得當前程式的環境變數
  - ✓ `execve(.., NULL)`
  - ✓ `execve(.., {NULL})`
  - ✓ `execve(.., environ)`

那麼哪裡找得到 execve 呢

```
system(cmd)
```

≈ fork() +

```
execve("/bin/sh", ["sh","-c",cmd], environ)
```

source code of `system`

```
pid = __fork ();
if (pid == (pid_t) 0)
{
  /* Child side.  */
  const char *new_argv[4];
  new_argv[0] = SHELL_NAME;
  new_argv[1] = "-c";
  new_argv[2] = line;
  new_argv[3] = NULL;

  /* ...omitted... */

  /* Exec the shell.  */
  (void) __execve (SHELL_PATH, (char *const *) new_argv, __environ);
  _exit (127);
```

直接跳在 system 中段

```
execve("/bin/sh", argv, environ)
```

↑

未初始化變數

如果 argv 或 *argv 是NULL

- ✓ `execve("/bin/sh", NULL, environ)`
- ✓ `execve("/bin/sh", {NULL}, environ)`

壞消息

One Gadget 不一定能成功 R

好消息

One Gadget 有很多個！

每個都試試總會中

那麼要怎麼找 One Gadget 呢

很久很久以前

大約半年以前

找 One Gadget 三步驟

(1)

搜尋 /bin/sh 字串的位置

```
➜ strings -tx /lib/x86_64-linux-gnu/libc.so.6 | grep /bin/sh
 18ac40 /bin/sh
```

&"/bin/sh": 0x18ac40

(2)

用 objdump 打開 libc.so.6

搜尋 `18ac40`

```
objdump -d libc.so.6|grep 18ac40 -A 7 -B 3
```

```
45575:      call   35a60 <sigprocmask@@GLIBC_2.2.5>
4557a:      mov    rax,QWORD PTR [rip+0x37b937]
45581:      lea    rdi,[rip+0x1456b8]        # 18ac40
45588:      lea    rsi,[rsp+0x30]
4558d:      mov    DWORD PTR [rip+0x37dee9],0x0
45597:      mov    DWORD PTR [rip+0x37dee3],0x0
455a1:      mov    rdx,QWORD PTR [rax]
455a4:      call   cd760 <execve@@GLIBC_2.2.5>
455a9:      mov    edi,0x7f
```

(3)

肉眼瞧瞧哪裡最像

```
execve("/bin/sh", argv, environ)
```

```
45575:      call   35a60 <sigprocmask@@GLIBC_2.2.5>
4557a:      mov    rax,QWORD PTR [rip+0x37b937]
45581:      lea    rdi,[rip+0x1456b8]        # 18ac40
45588:      lea    rsi,[rsp+0x30]
4558d:      mov    DWORD PTR [rip+0x37dee9],0x0
45597:      mov    DWORD PTR [rip+0x37dee3],0x0
455a1:      mov    rdx,QWORD PTR [rax]
455a4:      call   cd760 <execve@@GLIBC_2.2.5>
455a9:      mov    edi,0x7f
```

One Gadget: 0x4557a

execve("/bin/sh", rsp+0x30, environ)

找起來是滿簡單

但每次比賽都重找一遍

永遠不記得上次找的放哪去了

很低能

於是在很久以前

大約半年前

決定寫個工具自動化

github: david942j/one_gadget

GitHub, Inc. [US] | https://github.com/david942j/one_gadget

Features Business Explore Marketplace Pricing This repository Search Sign in or Sign up

david942j / one_gadget

Watch 10 Star 171 Fork 22

Code Issues 1 Pull requests 0 Projects 0 Insights

A tool for you easy to find the one gadget RCE in libc.so.6

ctf pwnable pwn glibc one-gadget-rce shell

102 commits 1 branch 16 releases 2 contributors MIT

Branch: master New pull request Find file Clone or download

david942j bump 1.4.0 Latest commit 9a9d6e2 on 23 Jun

bin	add gem version option	4 months ago
examples	update README.md	6 months ago
lib	bump 1.4.0	2 months ago
spec	show warning message when BuildID not found	2 months ago
tasks	Refactor/naming (#10)	2 months ago
.codeclimate.yml	Testing on appveyor (#7)	3 months ago
.gitignore	docs	5 months ago
.rubocop.yml	Fix/pltgot (#9)	2 months ago

目標

尋找會呼叫

```
execve("/bin/sh", argv, envp)
```

的程式片段

One Gadget 成功條件

- `argv` 與 `envp` 為合法 array of pointers
- 一般來說就是

```
argv == NULL || *argv == NULL
envp == NULL || *envp == NULL || envp == environ
```

工欲善其事 必先利其器

Symbolic Execution

分析有未知數時程式會如何執行

1 分鐘學會 Symbolic Execution

預備

開始

簡單地說

在有未知數的情境下執行程式

執行時

遇到未知變數就讓他是變數

```
x = input()
```

遇到運算都用變數表達

$$y = x*2 \rightarrow y = 2x$$

$$z = y*3-2 \rightarrow z = 6x-2$$

遇到分支就記錄限制條件

```
if z <= 10
```

constraint: `6x-2 <= 10`

```
if y <= z
```

constraint: `2x <= 6x-2`

合起來

```
x = input()
y = x * 2
z = y * 3 - 2
if z <= 10 and y <= z
  return SUCCESS
else
  return FAIL
```

解限制：`6x-2 <= 10 && 2x <= 6x-2`

`0.5 <= x <= 2` → `SUCCESS`

otherwise → `FAIL`

現在還剩 15 秒

總之

有變數地執行程式

遇到分支就新增限制條件

解限制 → 未知數多少時走到哪

這 94 Symbolic Execution

Symbolic Execution and One Gadget

目的

健康地找到 One Gadget

方法

找個起點 直接跑下去

所有 registers 都是變數

跑下去直到

```
call execve("/bin/sh", argv, envp)
```

One Gadget 成功條件

`argv` / `envp` 皆合法

舉個例子

ubuntu 16.04 32bit 函式庫片段

```
3ac69:   mov    eax,DWORD PTR [esi-0xb8]
3ac6f:   add    esp,0xc
3ac72:   mov    DWORD PTR [esi+0x1620],0x0
3ac7c:   mov    DWORD PTR [esi+0x1624],0x0
3ac86:   push   DWORD PTR [eax] ; environ
3ac88:   lea    eax,[esp+0x2c]
3ac8c:   push   eax
3ac8d:   lea    eax,[esi-0x567d5] ; "/bin/sh"
3ac93:   push   eax
3ac94:   call   0xb0670 <execve>
```

前情提要一下

i386 的 calling convention

- 呼叫函式時的參數是放在 stack 上
- `esp` 是 stack pointer

call func(A, B, C)

```
                                        esp (stack 的頭)
                                         |
+-----------+-----------+-----------V-----------------------
|     C     |     B     |     A     |
+-----------+-----------+-----------+-----------------------
```

ubuntu 16.04 32bit 函式庫片段

```
3ac69:   mov    eax,DWORD PTR [esi-0xb8]
3ac6f:   add    esp,0xc
3ac72:   mov    DWORD PTR [esi+0x1620],0x0
3ac7c:   mov    DWORD PTR [esi+0x1624],0x0
3ac86:   push   DWORD PTR [eax] ; environ
3ac88:   lea    eax,[esp+0x2c]
3ac8c:   push   eax
3ac8d:   lea    eax,[esi-0x567d5] ; "/bin/sh"
3ac93:   push   eax
3ac94:   call   0xb0670 <execve>
```

```
                                    esp
                                     |
+-----------+-----------+-----------V-----------------------
|           |           |           |
+-----------+-----------+-----------+-----------------------
```

esp += 0xc

```
3ac69:   mov    eax,DWORD PTR [esi-0xb8]
3ac6f:   add    esp,0xc
3ac72:   mov    DWORD PTR [esi+0x1620],0x0
3ac7c:   mov    DWORD PTR [esi+0x1624],0x0
3ac86:   push   DWORD PTR [eax] ; environ
3ac88:   lea    eax,[esp+0x2c]
3ac8c:   push   eax
3ac8d:   lea    eax,[esi-0x567d5] ; "/bin/sh"
3ac93:   push   eax
3ac94:   call   0xb0670 <execve>
```

esp = esp + 0xc

```
esp                                   esp
 |                                     |
 V-----------+-----------+-----------V-----------------------
 |           |           |           |
 +-----------+-----------+-----------+----------------------
```

push environ

```
3ac69:   mov    eax,DWORD PTR [esi-0xb8]
3ac6f:   add    esp,0xc
3ac72:   mov    DWORD PTR [esi+0x1620],0x0
3ac7c:   mov    DWORD PTR [esi+0x1624],0x0
3ac86:   push   DWORD PTR [eax] ; environ
3ac88:   lea    eax,[esp+0x2c]
3ac8c:   push   eax
3ac8d:   lea    eax,[esi-0x567d5] ; "/bin/sh"
3ac93:   push   eax
3ac94:   call   0xb0670 <execve>
```

esp = esp + 8

```
            esp                        esp
             |                          |
+-----------V-----------+-----------V-----------------------
|  environ  |           |           |
+-----------+-----------+-----------+-----------------------
```

eax = esp+0x2c

```
3ac69:   mov    eax,DWORD PTR [esi-0xb8]
3ac6f:   add    esp,0xc
3ac72:   mov    DWORD PTR [esi+0x1620],0x0
3ac7c:   mov    DWORD PTR [esi+0x1624],0x0
3ac86:   push   DWORD PTR [eax] ; environ
3ac88:   lea    eax,[esp+0x2c]
3ac8c:   push   eax
3ac8d:   lea    eax,[esi-0x567d5] ; "/bin/sh"
3ac93:   push   eax
3ac94:   call   0xb0670 <execve>
```

eax = esp + 0x34

```
            esp                        esp
             |                          |
+-----------V-----------+-----------V-----------------------
|  environ  |           |           |
+-----------+-----------+-----------+-----------------------
```

push eax

```
3ac69:   mov    eax,DWORD PTR [esi-0xb8]
3ac6f:   add    esp,0xc
3ac72:   mov    DWORD PTR [esi+0x1620],0x0
3ac7c:   mov    DWORD PTR [esi+0x1624],0x0
3ac86:   push   DWORD PTR [eax] ; environ
3ac88:   lea    eax,[esp+0x2c]
3ac8c:   push   eax
3ac8d:   lea    eax,[esi-0x567d5] ; "/bin/sh"
3ac93:   push   eax
3ac94:   call   0xb0670 <execve>
```

```
                        esp         esp
                         |           |
+-----------+-----------V-----------V-----------------------
|  environ  |  esp+0x34 |           |
+-----------+-----------+-----------+-----------------------
```

eax = &"/bin/sh"

```
3ac69:   mov    eax,DWORD PTR [esi-0xb8]
3ac6f:   add    esp,0xc
3ac72:   mov    DWORD PTR [esi+0x1620],0x0
3ac7c:   mov    DWORD PTR [esi+0x1624],0x0
3ac86:   push   DWORD PTR [eax] ; environ
3ac88:   lea    eax,[esp+0x2c]
3ac8c:   push   eax
3ac8d:   lea    eax,[esi-0x567d5] ; "/bin/sh"
3ac93:   push   eax
3ac94:   call   0xb0670 <execve>
```

```
                        esp          esp
                         |            |
+-----------+-----------V-----------V-----------------------
|  environ  |  esp+0x34 |            |
+-----------+-----------+-----------+-----------------------
```

push eax

```
3ac69:   mov    eax,DWORD PTR [esi-0xb8]
3ac6f:   add    esp,0xc
3ac72:   mov    DWORD PTR [esi+0x1620],0x0
3ac7c:   mov    DWORD PTR [esi+0x1624],0x0
3ac86:   push   DWORD PTR [eax] ; environ
3ac88:   lea    eax,[esp+0x2c]
3ac8c:   push   eax
3ac8d:   lea    eax,[esi-0x567d5] ; "/bin/sh"
3ac93:   push   eax
3ac94:   call   0xb0670 <execve>
```

```
                                        esp
                                         |
+-----------+-----------+-----------V-----------------------
|  environ  |  esp+0x34 | "/bin/sh" |
+-----------+-----------+-----------+----------------------
```

```
3ac69:   mov    eax,DWORD PTR [esi-0xb8]
3ac6f:   add    esp,0xc
3ac72:   mov    DWORD PTR [esi+0x1620],0x0
3ac7c:   mov    DWORD PTR [esi+0x1624],0x0
3ac86:   push   DWORD PTR [eax] ; environ
3ac88:   lea    eax,[esp+0x2c]
3ac8c:   push   eax
3ac8d:   lea    eax,[esi-0x567d5] ; "/bin/sh"
3ac93:   push   eax
3ac94:   call   0xb0670 <execve>
```

execve("/bin/sh", esp+0x34, environ)

```
                                        esp
                                         |
+-----------+-----------+-----------V-----------------------
|  environ  |  esp+0x34 | "/bin/sh" |
+-----------+-----------+-----------+-----------------------
```

去蕪存菁

條件重複的丟掉

✗ `*(esp+0x34) == NULL && eax == NULL`

✓ `*(esp+0x34) == NULL && eax == NULL`

成功條件較嚴苛的不要

✗ `*(esp+0x34) == NULL && eax == NULL`

✓ `*(esp+0x34) == NULL`

寫自動化工具以前

以為 OneGadget 只有三個

寫工具以後找到 5~8 個

Demo with a CTF challenge

Conclusion

結語

1. 身為工程師就是要把瑣事自動化
2. 用過 One Gadget 才瞭解有多好用
3. Symbolic Execution 還有很多細節喇
4. 喜歡就幫 One Gadget 按個 star 喇
   - https://github.com/david942j/one_gadget

david942j @