# Infecting Mach-O Files

roy g biv
*October 2006*

[Back to index]

MachoMan virus

## What is a Mach-O file?

Mach-O is the native file format used by OSX. There is a little similarity to Portable Executable files, but not much. Mach-O files are collections of segments. Each segment can contain one or more sections, which have different protection attributes.

## What does a Mach-O file look like?

Everything about the format is public, most of the format is in loader.h. The file header structure is called mach_header. Each of the fields is 32-bits large. It has this format:

| Offset | Field | Description |
|---|---|---|
| 0x00 | magic | sig (0xfeedface (PowerPC), 0xcefaedfe (Intel)) |
| 0x04 | cputype | 0x12 (PowerPC), 0x07 (Intel) |
| 0x08 | cpusubtype | specific architecture |
| 0x0c | filetype | 0x02 if executable |
| 0x10 | ncmds | number of commands following |
| 0x14 | sizeofcmds | total size of commands |
| 0x18 | flags | |

The commands are used for many different purposes, such as describing segments and sections, initial values of the CPU registers for the main thread, and resolving symbols (equivalent to imports in PE files).

The load_command structure has this format:

| Offset | Field | Description |
|---|---|---|
| 0x00 | cmd | type of command |
| 0x04 | cmdsize | number of bytes in command (the value here can be larger than the command data, so this field must be used to reach the next command, do not rely on the command data) |

Interesting commands are LC_SEGMENT (1) and LC_UNIXTHREAD (5). The LC_SEGMENT command describes a segment of memory. It is equivalent to a section in PE files. The segment_command structure has this format:

| Offset | Size | Field | Description |
|---|---|---|---|
| 0x00 | 16 | segname | name of segment (ignored, just like PE) |
| 0x10 | 4 | vmaddr | segment *virtual* address |
| 0x14 | 4 | vmsize | segment virtual size |
| 0x18 | 4 | fileoff | segment file offset |
| 0x1c | 4 | filesize | segment file size (0 means empty) |
| 0x20 | 4 | maxprot | maximum protection attributes (can disallows writable code, for example, but clearing PROT_WRITE bit) |
| 0x24 | 4 | initprot | initial protection attributes (combination of READ, WRITE, EXEC, but PROT_WRITE requires PROT_READ) |
| 0x28 | 4 | nsects | number of sections following |
| 0x2c | 4 | flags | |

A section is a piece of memory within a segment. The section_command structure has this format:

| Offset | Size | Field | Description |
|---|---|---|---|
| 0x00 | 16 | sectname | name of section |
| 0x10 | 16 | segname | name of host segment |
| 0x20 | 4 | addr | section virtual address |
| 0x24 | 4 | size | section file size |
| 0x28 | 4 | offset | section file offset |
| 0x2c | 4 | align | section alignment |
| 0x30 | 4 | reloff | relocation data file offset |
| 0x34 | 4 | nreloc | relocation data item count |
| 0x38 | 4 | flags | |
| 0x3c | 4 | reserved1 | interpretation depends on flags |
| 0x40 | 4 | reserved2 | interpretation depends on flags |

The flags are a packed structure, the low 8 bits describe the section type, the top 8 bits describe the section user attributes, the next 8 bits describe the section system attributes.

## How do we infect it?

I thought about this problem for a long time. The problem with the format is that some structures, like the symbol tables access sections by number, so we can't insert sections or segments. We could add a section to the end, but that would require possibly moving file data to make room, and some structures are difficult to parse properly, so that's not a good option. I thought about a cavity infector, but the only good cavity that I could find was in the __jump_table section, but the size cannot be altered, because it is used by the symbol loader. I considered appending to the __LINKEDIT segment, but it is discarded by the loader. I thought about moving some code from the __text section to the end of the file, and placing myself in the space, but then I would need to open the file to read it back.

Eventually, I started thinking about it differently. Each file is supposed to start with a __PAGEZERO segment, which marks the first 0x1000 bytes as not accessible. The file size there is 0, but I wondered if I could change it and load my code? Amazingly, it is so. All I had to

do was pad the file to a multiple of 4kb first, to avoid a bus error, then append my code. After that, I set the file offset and size fields, and the protection flags so I can run.

## How to get control?

This was a problem, too, for some time. I was using IDA to load the file, but at first I didn't see anywhere the entrypoint value. It seems that the Ilfak had the same problem, because IDA assumes that the entrypoint is always the first byte in the __text section. Of course, that's not true. :)

## Introducing LC_UNIXTHREAD

The LC_UNIXTHREAD load command describes the register values for the main thread in the file. Yes, that includes EIP. By simply changing the value in the EIP field to another value, I was able to move the entrypoint around, but IDA did not notice, and continued to show the old one! It's a new type of entrypoint obscuring. ;) Even more interesting was that IDA refuses to load any segment which contains no sections (like __LINKEDIT and, more importantly, __PAGEZERO). That means my code is invisible, yet it runs.

The structure is of the thread_command type. It has this format:

| Offset | Size | Field | Description |
|---|---|---|---|
| 0x00 | 4 | flavor | type of data following |
| 0x04 | 4 | count | number of dwords following |

The interpretation of the thread information depends on the data flavor. We are interested only in the i386_NEW_THREAD_STATE (1). In that case, it is a i386_thread_state_t structure, and it has the format:

| Offset | Size | Field |
|---|---|---|
| 0x00 | 4 | eax |
| 0x04 | 4 | ebx |
| 0x08 | 4 | ecx |
| 0x0c | 4 | edx |
| 0x10 | 4 | edi |
| 0x14 | 4 | esi |
| 0x18 | 4 | ebp |
| 0x1c | 4 | esp |
| 0x20 | 4 | ss |
| 0x24 | 4 | eflags |
| 0x28 | 4 | eip |
| 0x2c | 4 | cs |
| 0x30 | 4 | ds |
| 0x34 | 4 | es |
| 0x38 | 4 | fs |
| 0x3c | 4 | gs |

and then we are done.

Greets to friendly people (A-Z):

Active - Benny - Malum - Obleak - Prototype - Ratter - Ronin - RT Fishel - sars - SPTH - The Gingerbread Man - Ultras - uNdErX - Vallez - Vecna - VirusBuster - Whitehead


```
rgb/defjam oct 2006
iam_rgb@hotmail.com
```


[Back to index]