seriot.ch

Beauty will save the world.
Fyodor Dostoevsky

Home
Home Made Maps
Hello Mach-O
A Tiny NTP Client
Abusing Twitter API
Data Visualization
Talks and Papers
Contact

Popular resources

Unicode Talk
Unicode Hacks
Unicode Poster
Abusing Twitter API
Twitter API Visual Doc.
Hello Mach-O
iPhone Privacy Talk
Quickies

External profiles

Twitter
LinkedIn
Facebook
GitHub
StackOverflow
Delicious
Amazon Wishlist

Hello Mach-O

Dissection of minimal Intel 32-bits, 204 bytes, Mach-O "Hello World" executable file.
December 2012 / January 2013

Writing a Minimal Mach-O Executable File

I am a big fan of the Corkami web site by Ange Albertini. I especially like his Portable Executable 101 poster. I wondered what it would take to describe a Mach-O executable file for Mac OS X.

Here is the way I took:

1. I modified the "Hello world" assembly program hello.asm by Peter Michaux and stored the string into `.text` instead of `.data` segment. The goal was to use only one segment to keep the executable file as simple as possible. I then compiled my own hello.asm into an object file with NASM.
2. I created the exectable file from `hello.o` with `ld` and options to reduce the executable complexity, namely `-static -pagezero_size 0 -no_uuid`.
3. I removed the `__DATA`, `__LINKEDIT` segments and the `LC_SYMTAB` load command. I also removed the symbol table and the string table, since they were useless.
4. I relocated by hand the `__TEXT` segment so that the result would be as small as possible.
5. [2013-01-01] I removed the `__TEXT.__text` section as suggested by @shantonusen.
6. [2013-01-02] Whenever possible, I optimized opcodes for size, as suggested by @ange4771.

The file is now 204 bytes long. It contains one single text segment. It does not use `printf` to avoid linking with libraries, it does write to stdout with a syscall instead. I also removed all the padding zeros.

For sure, such a simple file is far from what you will find in the real-world, but analyzing how it works is a nice way to get insight on the Mach-O file format and the Mac OS X loader.

For the record, the tools I used are Hex Fiend, MachOView, nasm, otool, ld and xxd.

The File

Download the file here: hello.zip

```
$ shasum hello
29866d22f3c262eb1ac96f520f78559311875281
```

Here is the file dumped by `xxd`. From left to right, we can see the offset address, the actual bytes (groupped by two) and their ASCII representations.

```
$ cat hello.hex
0000000: cefa edfe 0700 0000 0300 0000 0200 0000  ................
0000010: 0200 0000 8800 0000 0100 0000 0100 0000  ................
0000020: 3800 0000 5f5f 5445 5854 0000 0000 0000  8...__TEXT......
0000030: 0000 0000 0000 0000 0010 0000 0000 0000  ................
0000040: 4000 0000 0700 0000 0500 0000 0000 0000  @...............
0000050: 0000 0000 0500 0000 5000 0000 0100 0000  ........P.......
0000060: 1000 0000 0000 0000 0000 0000 0000 0000  ................
0000070: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000080: 0000 0000 0000 0000 0000 0000 a400 0000  ................
0000090: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000a0: 0000 0000 6a0c 68c0 0000 006a 01b0 0483  ....j.h....j....
00000b0: ec04 cd80 83c4 106a 00b0 0183 ec04 cd80  .......j........
00000c0: 4865 6c6c 6f20 776f 726c 640a            Hello world.
```

You can save this file and convert it back to binary with `xxd`:

```
$ xxd -r hello.hex > hello
```

Now we just need to make it executable...

```
$ chmod +x hello
```

...and the file can be run.

```
$ ./hello
Hello world
```

So, as you can see, there is nothing more than these 204 bytes.

Ange Albertini wrote an assembly source file for a slightly different version of this file: helloworld.asm. You can use it like this:

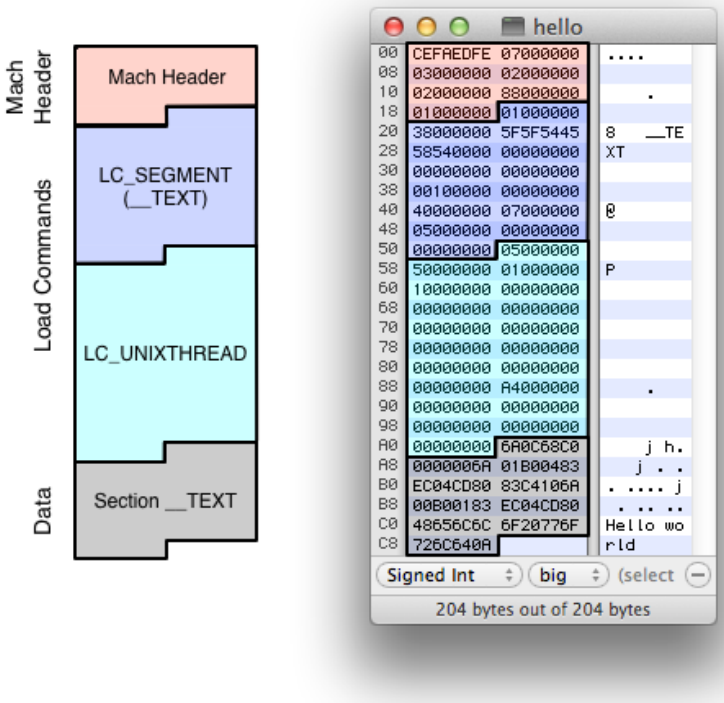
```
$ nasm -f bin helloworld.asm
$ chmod +x helloworld
$ ./helloworld
Hello world
$ shasum helloworld
64fc36aa88aa0403a3d276466a7aac47d106f490  helloworld
```

High Level View

Now if we what to know what the bytes mean, we have to read the OS X ABI Mach-O File Format Reference. We read that Mach-O files contain three main parts:

- the header - it describes the kind of Mach-O file
- load commands - it describes the different segments to be loaded in memory and their sections
- segment sections data - the actual data, such as constants and executable code

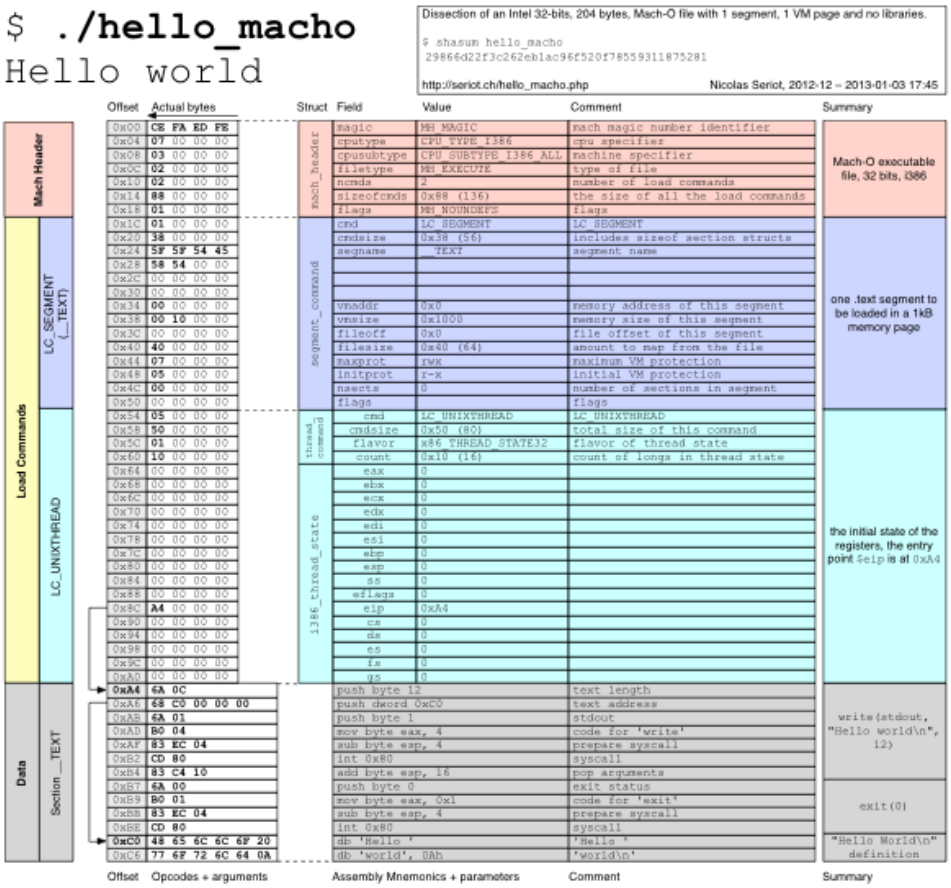
We can see these three main parts in our hello Mach-O executable.



File Dissection

In order to explain the exact meaning of all the 204 bytes, here is another view of the same file, 4 bytes (32 bits) at a time.

Click to get a full-scale PDF file.



[2013-01-04] A Hacky but Valid Micro "Hello world" Macho-O File

Looking for valid ("not crashing" and "not raising issues from `otool -l`") minimal Mach-O files on the Internet yields:

- 248 bytes, nicertiny.asm by Amit Singh, return status 42
- 164 bytes, tiny_mfeiri.asm by Michael Feiri, return status 42
- 242 bytes, tine_hello.s (64 bits), softboysxp, prints `Hello, World!`
- 204 bytes, hello_macho, Nicolas Seriot (myself), prints `Hello world`
- 180 bytes, rev_mach-o.asm, Alfredo Pesoli (@__rev), prints `Hello world`

Let me add another stone to the garden and introduce `micro_macho`:

- 164 bytes, `micro_macho`, prints `Hello world`

Download micro_macho.zip or use the hex dump as follows:

```
$ cat micro_macho.hex
0000000: cefa edfe 0700 0000 0300 0000 0200 0000  ................
0000010: 0200 0000 8800 0000 0100 0000 0100 0000  ................
0000020: 3800 0000 4865 6c6c 6f20 776f 726c 640a  8...Hello world.
0000030: 00ff ffff 0000 0000 0010 0000 0000 0000  ................
0000040: 2e00 0000 07ff ffff 05ff ffff 0000 0000  ................
0000050: ffff ffff 0500 0000 5000 0000 0100 0000  ........P.......
0000060: 1000 0000 ff00 ffff 6a0c 6824 0000 006a  ........j.h$...j
0000070: 01b0 0483 ec04 cd80 83c4 106a 00eb 11ff  ...........j....
0000080: 0000 0000 ffff ffff ff00 ffff 6800 0000  ............h...
0000090: b001 83ec 04cd 80ff ffff ffff 0000 ffff  ................
00000a0: 0000 ffff                                ....

$ xxd -r micro_macho.hex > micro_macho

$ shasum micro_macho
e67bddcc7ba3f8446a63104108c2905f57baadbe  micro_macho

$ chmod +x micro_macho

$ ./micro_macho
Hello world
```

I proceeded by:

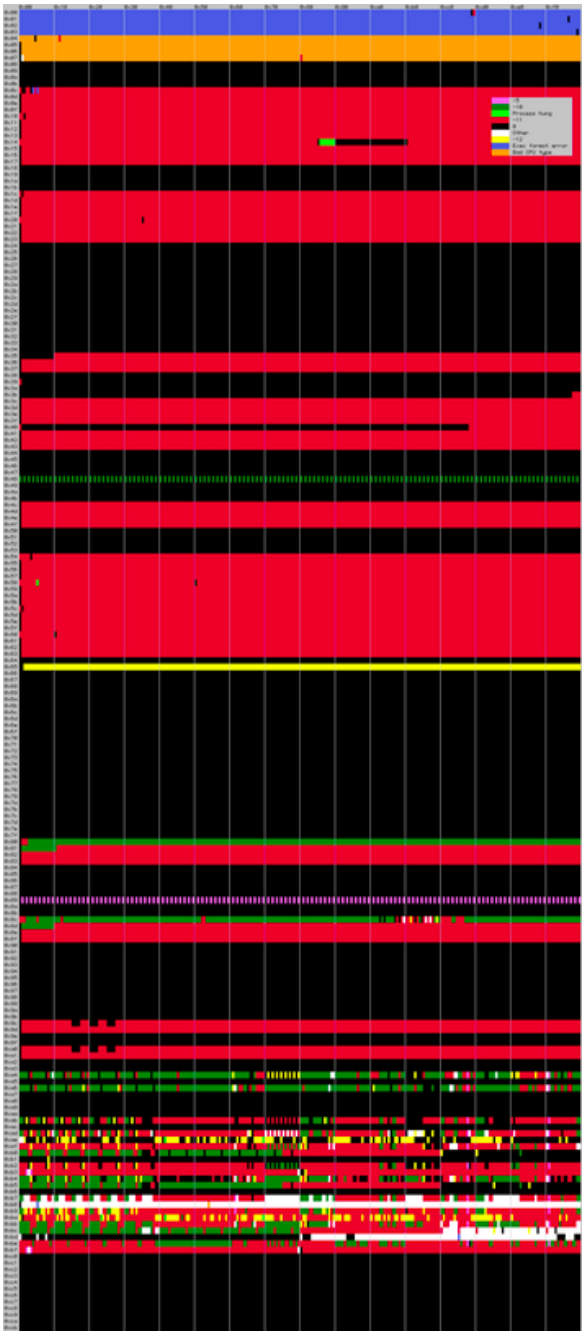
1. totally removing the `TEXT` section (as in tiny_mfeiri.asm)
2. fuzzing every byte to know what is used and what is not (see picture below)
3. overwrite free bytes with `FF` whenever possible
4. stuffing the string into `LC_SEGMENT.segname`

5. stuffing minimized opcodes into register initial states, jumping when necessary
6. updating the entry point `$eip` and the string address

Here is a quick and dirty visualization of the fuzzing return statuses.

One line per byte, one column per possible byte value, status color according to the (partial) legend.

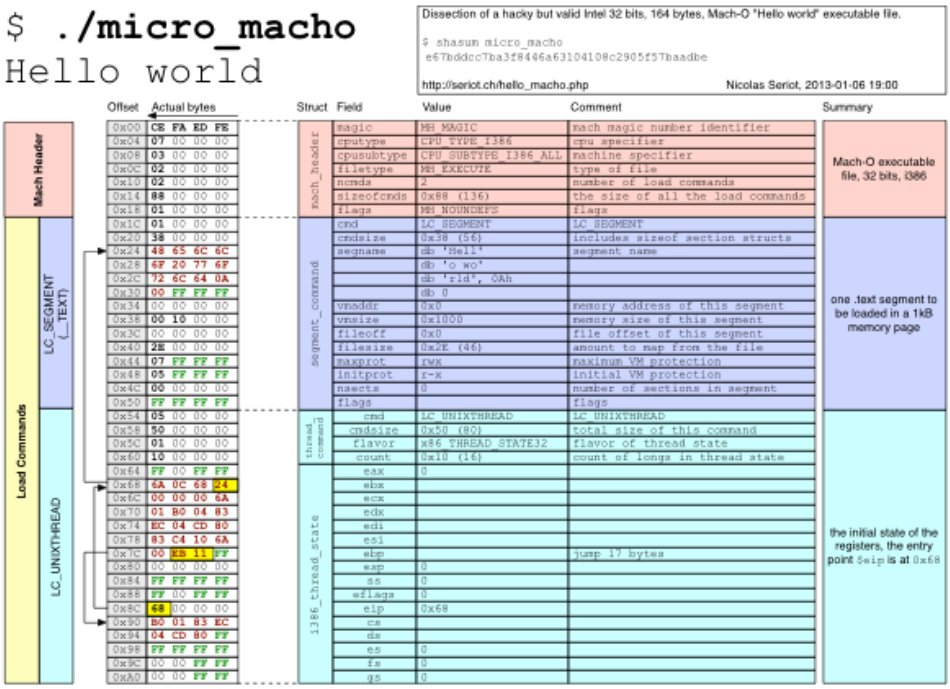
Black cells shows return status 0 (which does not imply that the string is printed correctly).



Now that we know which bytes we can reuse, we can stuff the executable code in them. Here is a visualisation of `micro_macho` in which all of this should be pretty obvious.

I highlighted the jumps and references in yellow, the "stuffed" bytes from the former `TEXT` section in red and the remaining `FF` free bytes in green.

Click to get a full-scale PDF file.



<Amit Singh mode>
There are still plenty of FF (and zeros) lurking in there! :-)
</Amit Singh mode>

Acknowledgments

Ange Albertini, Shantonu Sen, Kevin Li

References

http://michaux.ca/articles/assembly-hello-world-for-os-x
http://osxbook.com/blog/2009/03/15/crafting-a-tiny-mach-o-executable/
http://feiri.de/macho/
http://www.0xcafebabe.it/2013/01/04/tiny-mach-0-are-fun/

/mach-o/loader.h
/osfmk/mach/i386/_structs.h
/bsd/kern/syscalls.master

Intel instruction set reference
X86 Opcode and Instruction Reference