sureSEC
SECURING THE SOURCE

Infecting the Mach-o Object Format

By Neil Archibald

Introduction

- Who am I? Neil Archibald, Senior Security Researcher @ Suresec Ltd
- Interested in Mac OSX sys-internals and research for roughly two years.
- Prior Knowledge of other object formats or assembly would be useful but is not necessarily required for this talk.
- Not intended to be a HOW-TO guide for Apple virus writers, but rather explore the Mach-o format and illustrate some ways in which infection can occur.

FelineMenace.org

suresec
SECURING THE SOURCE

Myth!

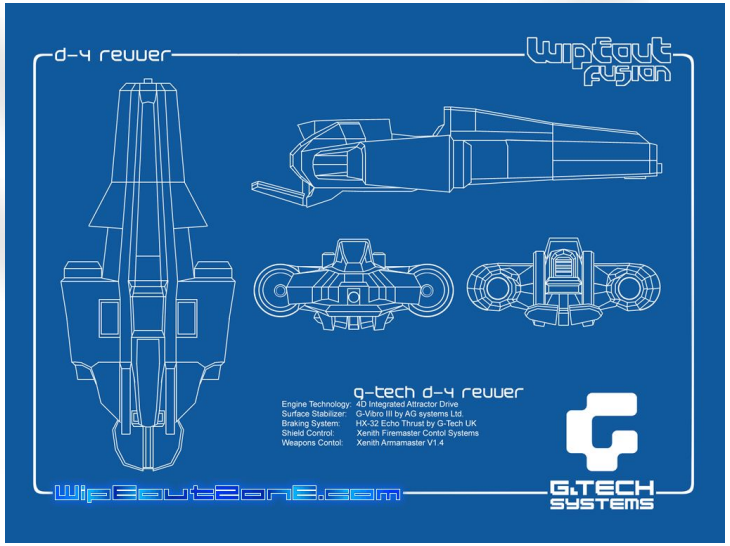
- Mac OSX is NOT immune to viruses or worms.

Infection

- Virus != Worm
- Infection is the process of injecting parasite code into a host binary.

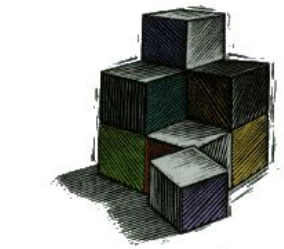
What is an Object Format?

- An object format is a file format used to store object code, and associated meta-data.
- It is like a blueprint which explains how to create a process image in memory.
- The object file itself is usually produced by a compiler or an assembler.
- There are many different object formats, these include ELF,PE-COFF and, of course, Mach-o.



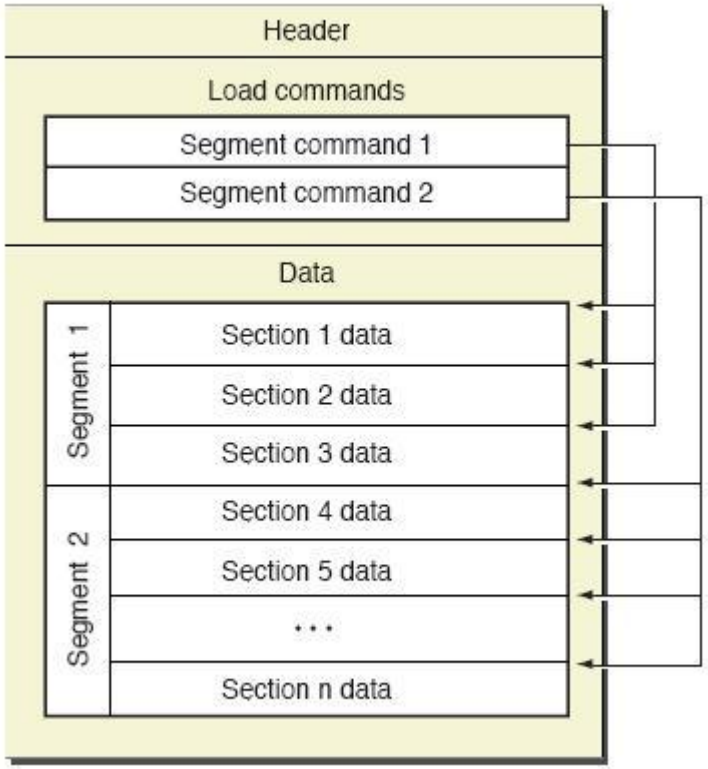
Introduction to Mach-o

- Object format used on operating systems which are based on the Mach kernel. (Such as NextStep and Mac OS X).
- Mach-o files are recognizable by the fact that all files begin with the four bytes *0xfeedface* as the magic number.



Mach-o Layout

- 3 main regions, header, load commands and sections.
- Each Segment command has 0 or more section commands associated with it.
- Sections are numbered starting from 1. Numbers continue increasing between different segments.



Mach-o Header

- Header structure found in */usr/include/mach-o/loader.h*
- Magic number as mentioned earlier is *0xfeedface*.
- CPU Information.
- File type, integer which specifies what kind of object file this it. Executable, shared library, bundle, etc.
- Combined size of all the load commands.
- Flags. Used to set various Mach-o options, such as binding style or a flat namespace.

```
struct mach_header
{
    uint32_t magic;
    cpu_type_t cputype;
    cpu_subtype_t cpusubtype;
    uint32_t filetype;
    uint32_t ncmds;
    uint32_t sizeofcmds;
    uint32_t flags;
};
```

Load Commands

- Each of the various load commands begin with the load_command struct.
- The command field specifies the type of each load command in the file.
- The most important of these to note at this stage are:

- *LC_SEGMENT*
- *LC_THREAD/LC_UNIXTHREAD*

```
struct load_command
{
    uint32_t cmd;
    uint32_t cmdsize;
};
```

LG_SEGMENT

- Specifies a portion of the file which is to be mapped into the address space of the process.
- Self explanatory fields size file offset, virtual size and virtual address.
- *nsects* represents the number of “section commands” which are associated with this *load_command*.
- Segments are usually named with a capital letter for clarity.
- An example of a Segment is the __TEXT segment.

```
struct segment_command
{
    uint32_t cmd;
    uint32_t cmdsize;
    char segname[16];
    uint32_t vmaddr;
    uint32_t vmsize;
    uint32_t fileoff;
    uint32_t filesize;
    vm_prot_t maxprot;
    vm_prot_t initprot;
    uint32_t nsects;
    uint32_t flags;
};
```

LC_THREAD

- Thread commands hold the initial state of the registers when a thread starts.
- This load_command can be used to retrieve of modify the entry-point for a thread.


```
struct thread_command
{
    uint32_t cmd;
    uint32_t cmdsize;
    /* uint32_t flavor;*/
    /* uint32_t count; */
    /* struct cpu_thread_state state;*/
};
```

Sections

- Sections have corresponding parent Segment commands.
- Multiple sections for one segment.
- They follow a lowercase naming convention.
- An example of a section is the __text section, which is part of the __TEXT segment.
- The most common flag setting used is S_REGULAR.
- **/usr/include/mach-o/loader.h** shows the possible options for flags.

```
struct section
{
    char sectname[16];
    char segname[16];
    uint32_t addr;
    uint32_t size;
    uint32_t offset;
    uint32_t align;
    uint32_t reloff;
    uint32_t nreloc;
    uint32_t flags;
    uint32_t reserved1;
    uint32_t reserved2;
};
```

Common Segment/Section Pairs

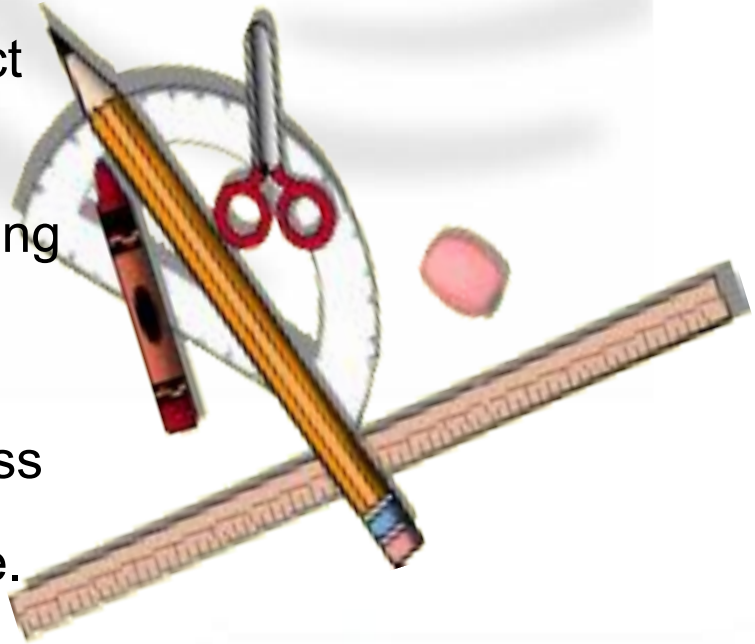
- **__TEXT,__text:** Generally stores executable machine code.
- **__DATA,__data:** Initialized variables are stored here.
- **__TEXT,__symbol_stub:** Used to store pieces of code which dereference and jump to a *lazy (or not) symbol pointer*. The dynamic linker fills in the pointer value.
- **__DATA,__la_symbol_ptr:** Lazy symbol pointer (mentioned above). **__DATA,__nl_symbol_ptr** == Not lazy.
- **__DATA,__bss:** Used to store uninitialized static variables.

Common S/S Pairs continued…

- **__DATA,__const:** Used to store relocatable constant variables.
- **__DATA,__mod_init_func**: Module constructors (similar to ctors) for C++.
- **__DATA,__mod_term_func:** Module destructors (similar to dtors) for C++.

Tools

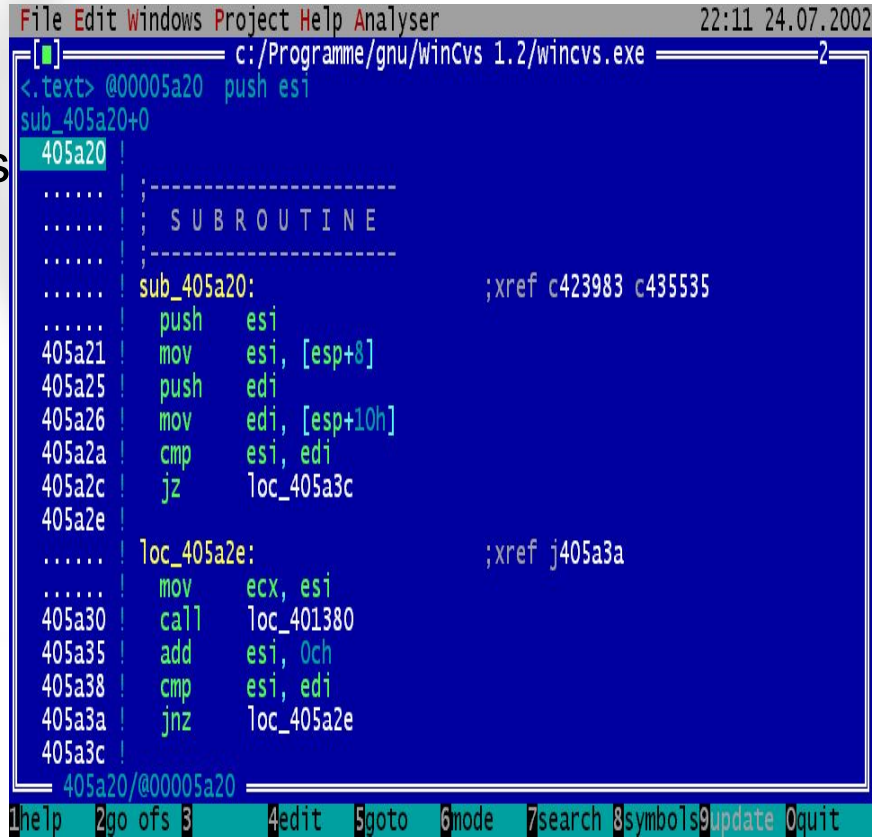
- **otool**: Kind of like **objdump** and **ldd**. Useful for dumping a disassembly of a file, libraries it's linked with. etc.
- **gdb**: Useful userland debugger.
- **gas**: gnu assembler.
- **libtool** (not gnu): Creating libraries (object files)
- **file**: Determining the type of a file.
- **ktrace:** A process tracer implemented using the *ktrace()* syscall which enables kernel logging for a process.
- **kdump:** Display the output from **ktrace**.
- **class-dump:** Display full Objective-C class listings of a mach-o file. Useful for reverse-engineering Objective C software.



HTE

- Free tool for manipulating object files.
- Makes changing object file headers trivial.
- Also supports code disassembly.
- Supports a variety of object file formats on various platforms.

- ***http://hte.sourceforge.net***



Concatenation method

- The first time I saw this was in b4b0 ezine, written up by Silvio Cesare.
- When you cat two executable objects together and run the result the original code will execute.

Concatenation method continued….

- To use this situation in order to an infect a file we can simply create a file which knows it's own size.
- When run, it must seek to the end of it's original self, and copy out the rest of the file into a temporary file to be executed.
- It must then execute whatever payload is desired, before executing the original, host binary for the user.

Concatenation method continued….

- Trivial to implement on Mac OS X.
- Process simply opens a file descriptor to it's own binary. (unlike Linux where /proc/pid/mem is used.) and reads the original file from it to /tmp.
- An implementation of this for mach-o/Mac OS X is online at http://felinemenace.org/~nemo/tools/mach-cat.tar.gz

Resource fork infection

- Mac OS X file system is called HFS+.
- Each file on a HFS+ partition has two file forks, a “data” fork and a “resource” fork.
- To access a files resource fork we can use **<filename>/rsrc**

Resource fork infection continued

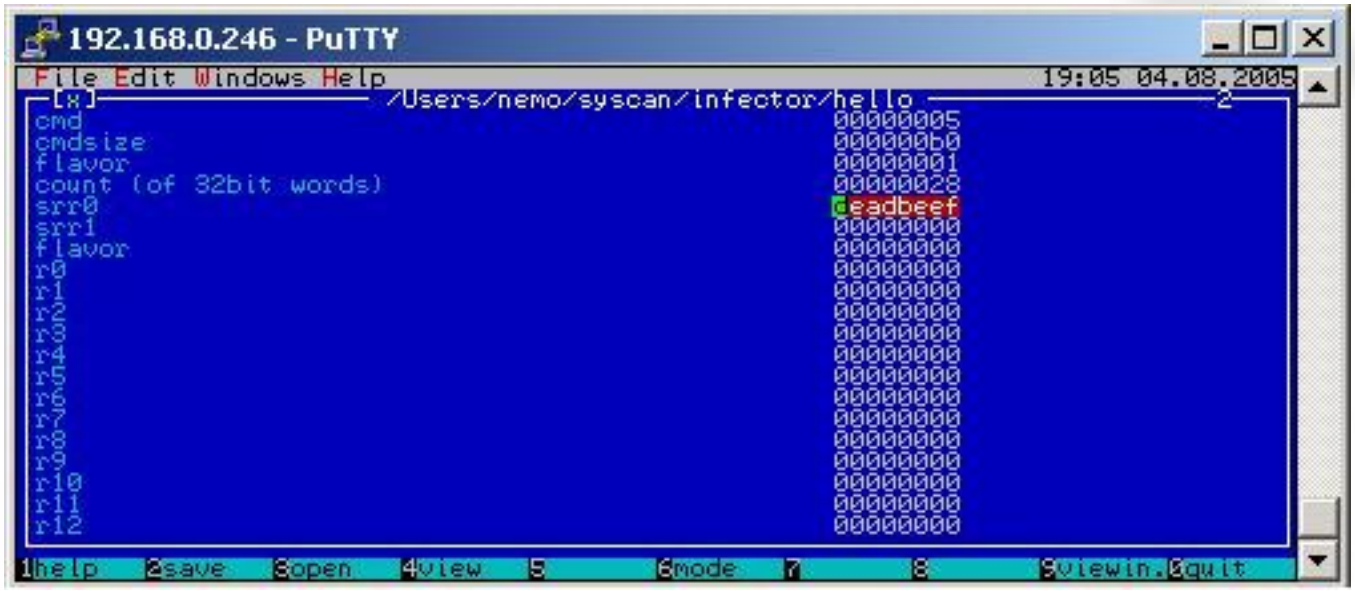
- To use this in order to infect a file, we can copy our host binary into the resource fork of our parasite.
- We then move the newly created file, over the existing host.
- When our new binary is executed, it simply *execve()* (executes) it's own resource fork after it's payload has completed.
- A problem with this however is that it will only work on the HFS file system. There is also talk that resource forks will be removed in the future.

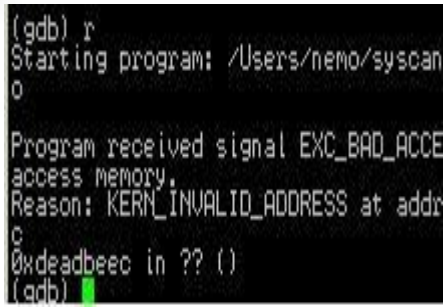
Resource fork infection continued

- My implementation of this technique is available online at: http://felinemenace.org/~nemo/tools/rsrc-hook.tar.gz

Thread entry point.

- The entry point for the initial thread can be found in a **LC_THREAD** or **LC_UNIXTHREAD** load command.
- The struct for this command contains an additional struct (**cpu_thread_state state**) which stores the initial state of each of the registers.
- The **srr0** field of this struct contains the entry point for the thread.
- The screenshot below shows HTE being used to modify the entry point of a binary.



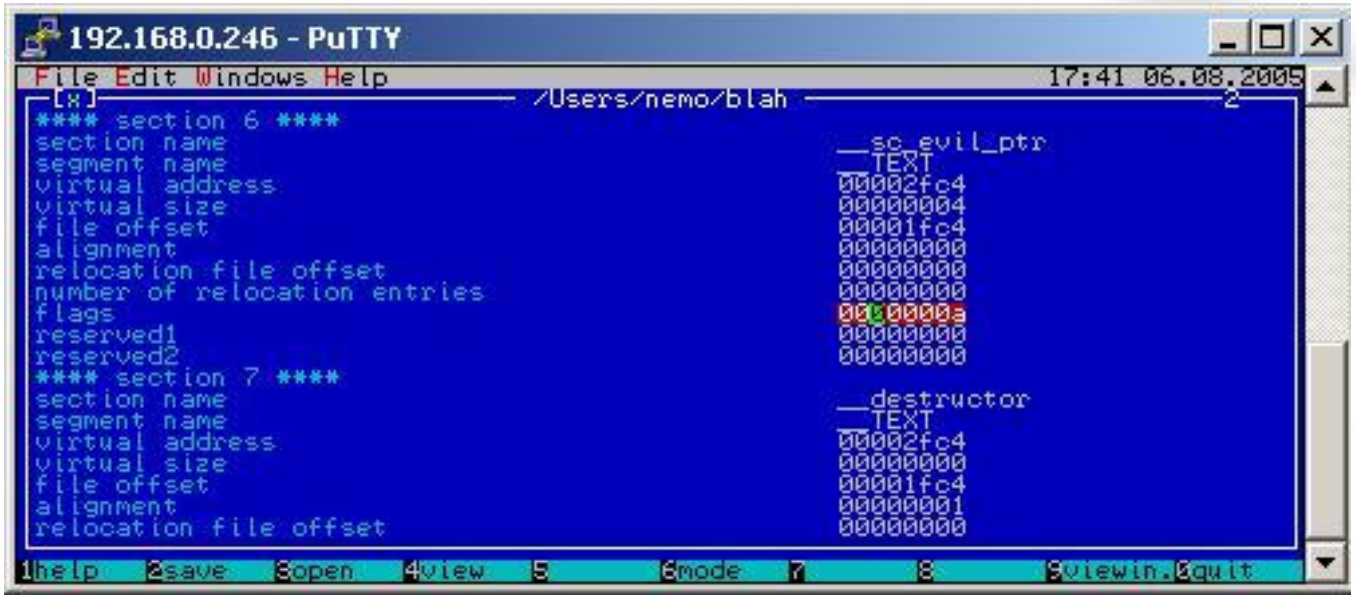


Alternate ways to hook entry-point

- Changing the entry point can easily be detected by anti-virus software.
- In some C++ applications, the *__DATA,__mod_init_func* section can be used to hook entry point.
- All c++ binaries compiled with g++ have a *__TEXT,__constructor and __TEXT,__destructor* sections. Even if they don't use it.
- We can use these sections to hook the entry point, or exit point, and also to store our code in memory.

A.W.T.H.E.P Continued…

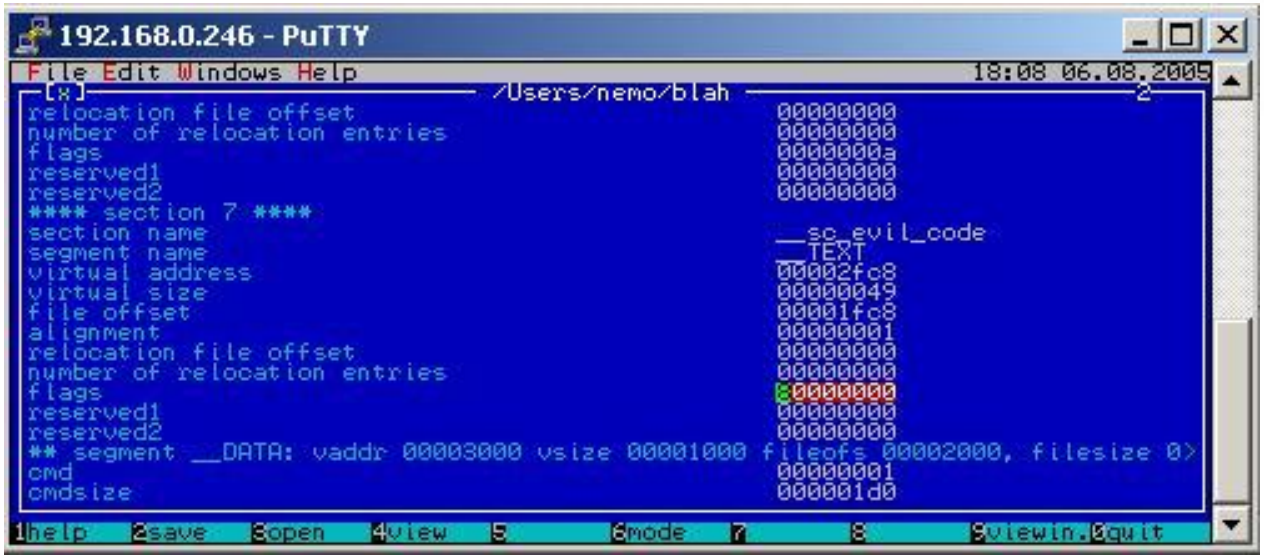
- Firstly we change the flags of the constructor to make it S_MOD_TERM_FUNCTION_POINTERS type. Marking the section this way means that it will be treated as a list of 4 byte addresses, to be called on program termination.
- After this we give this section size (4 bytes) to hold the address of our payload.



suresec
SECURING THE SOURCE

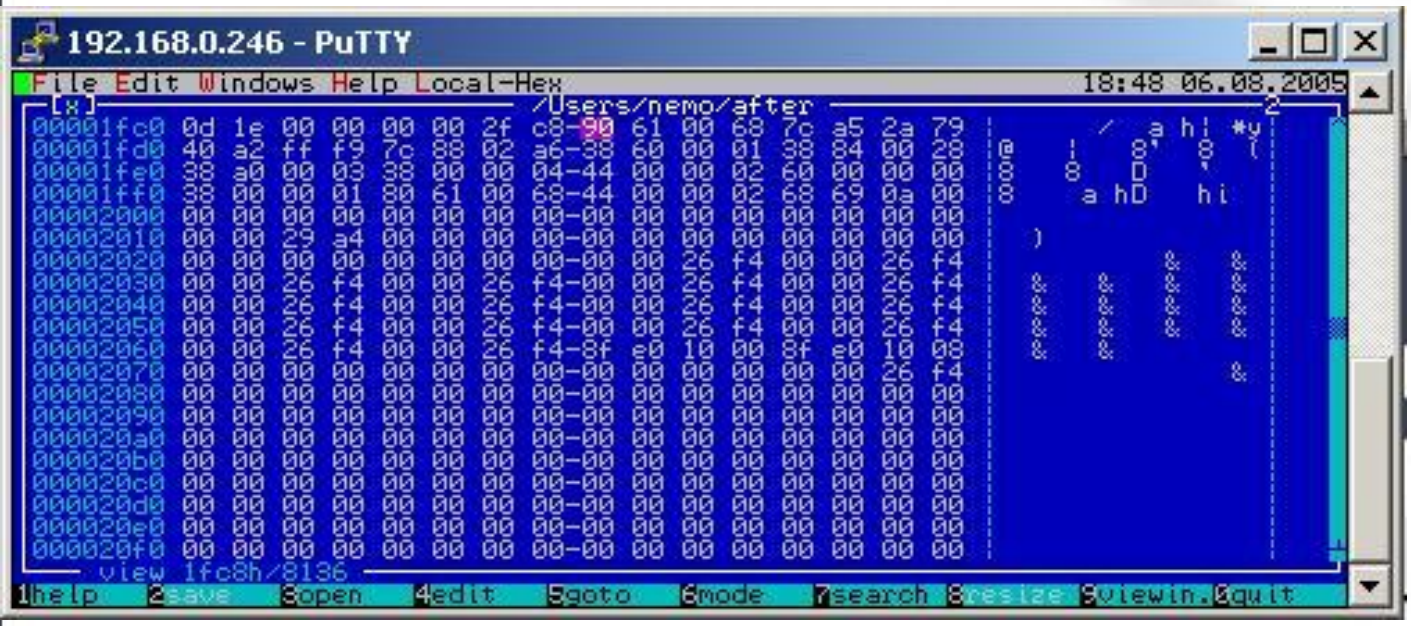
Storing code …

- Now that we have room for a pointer, which will be used to control execution on our binaries exit, we must make room for our code.
- To do this we can modify the destructor section of our binary and store our code there.
- We can change the virtual size of this section to be the size of our shellcode. In this case we will use simple write() shellcode.
- We must also modify the virtual address of this section. 4 bytes need to be added to the virtual address in order to make room for the pointer in the previous slide. This must also be done to the offset.
- Finally the “flags” field of our section must be set to 0x80000000 to indicate that executable code will be stored in this section.

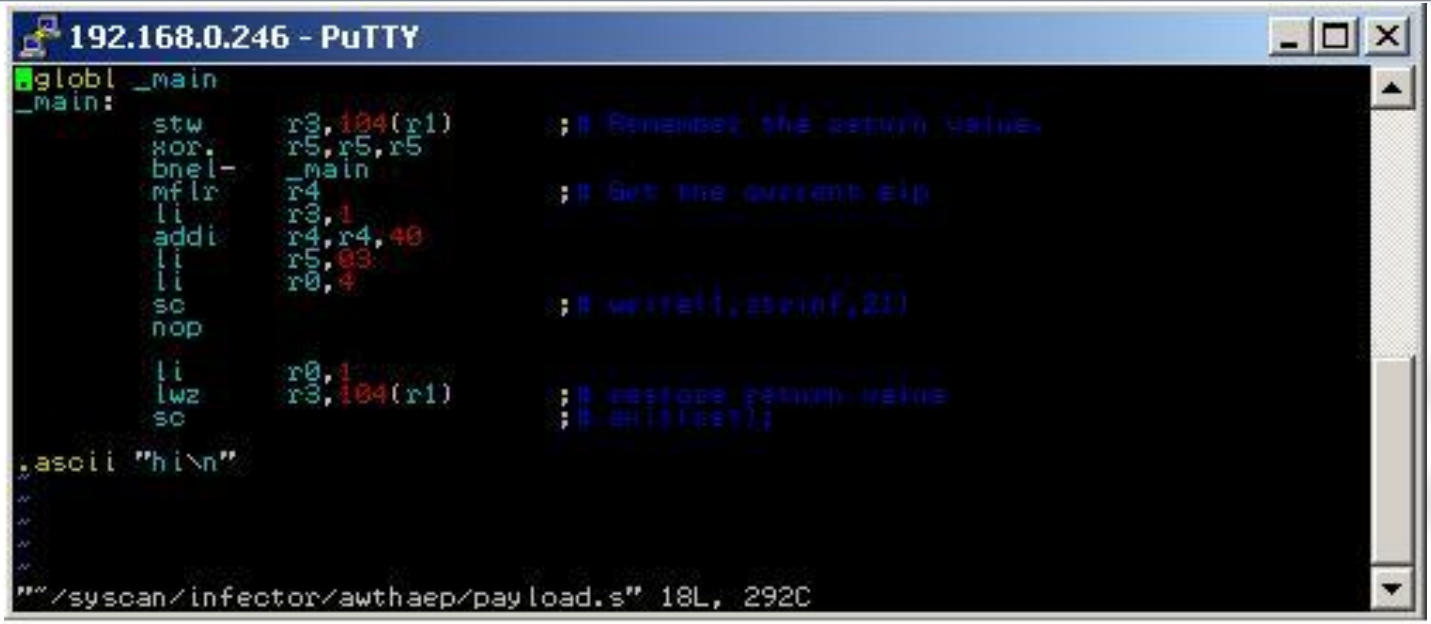


Storing code…

- Now that our headers have been set up, we need to actually copy the address of our code, followed by the code itself into the start of our new section.



Finished Infection

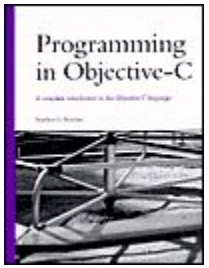


Kernel Infection

- Kernel extensions consist of an *.ext/ directory which contains meta-data and the kext (mach-o) binary (typically in Contents/MacOS/).
- The kernel itself is an uncompressed mach-o file as well. Unlike linux's kernel which is compressed.
- This allows for easy editing of the running kernel on disk, and in memory via /dev/kmem.

Objective-C Runtime Architecture

- Many of the larger applications on Mac OS X are written in a language called Objective C.
- Programs written in Objective C are typically linked with the Objective C runtime in */usr/lib/libobjc.A.dylib.*
- An __*OBJC* segment is added to the file in order to store data used by the Objective-C language runtime support library. Created by the Objective-C compiler.
- The **otool** tool can be used to view the contents of this segment (otool –vo <filename>). Also the **class-dump** tool displays this information in an easily read fashion.



Method Swizzling

- Method swizzling was pointed out to me by Braden Thomas. He wrote a paper and implementation showing how to use it in order to hook Mail.app.
- Method swizzling is the name given to the process of hooking an Objective-C method within a class.
- One of the functions of the __*OBJC* segment is to provide the Objective-C runtime with somewhere to store mappings between selectors (objective-c methods names) to the implementation of the method (actual program code.)
- These mappings can easily be modified in order to effectively hook a single method.

Method Swizzling continued…

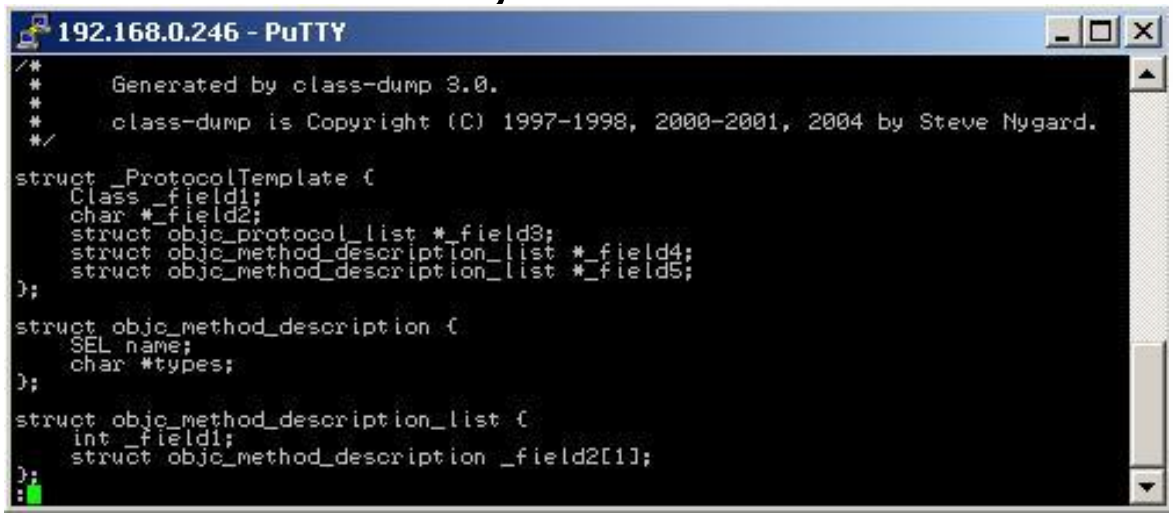
- The website: http://www.cocoadev.com/index.pl?MethodSwizzling shows an implementation of this which can easily be modified to perform Method Swizzling of any chosen method.
- In order to actually load the payload into memory Braden suggests the use of “InputManager”. Any bundles which are placed in the InputManager directory, in either the users Library directory, or the global Library directory, will be mapped in to every application opened.
- When combined with method swizzling, this provides an easy way to infect an application.

Class Posing

- Class posing is a “feature” of the objective-c runtime library.
- It allows you to replace an entire class with your own, in this way you can hook an entire class.
- An explanation of it is available here: http://www.cocoadev.com/index.pl?ClassPosing
- poseAsClass() function!

Infecting *libobjc.A.dylib*

- As mentioned earlier the ***libobjc.A.dylib*** library is linked with every program which is compiled with a Objective-C compiler.
- Due to the fact that this library is, itself written in Objective-C, we are able to use class-dump in order to locate key methods and swizzle them ourselves.
- In this way we can hook functions across all Objective-C binaries on the system.



Universal Binaries (FAT)

- Mac OS X moving to x86 from ppc.
- Need to support more than one architecture in a single file.
- Not really a mach-o file, but an archive containing mach-o files.

Infecting Universal Binaries

- Best method is to infect each of the files separately.
- Trivial format makes extracting the files for infection easy.
- *http://felinemenace.org/~nemo/tools/fm-unipack.tar.gz* - Tool for manipulating universal binaries. Listing contents and packing or unpacking universal binaries.

fat_header

- All FAT universal binaries begin with the fat_header struct.
- This struct consists of a magic number *0xcafebabe*, followed by the number of "*fat_arch*" structs which follow the header.
- Each "*fat_arch*" struct describes a single mach-o file within the universal binary.
- This struct is defined in the file: */usr/include/mach-o/fat.h*.

```
struct fat_header
{
    uint32_t magic;
    uint32_t nfat_arch;
};
```

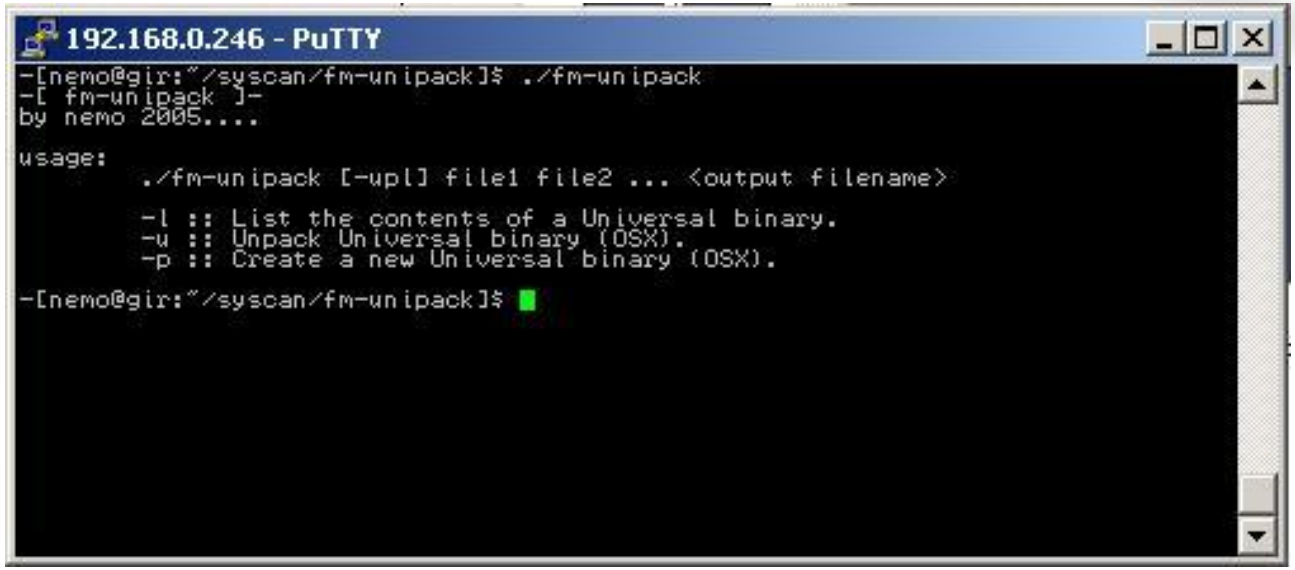
fat_arch

- Each fat_arch struct contains information about each of the files in the FAT binary.
- The first two fields show information about the type of architecture.
- The offset and size fields (obviously) are used to store information about the size and starting location of the file. Appropriate alignment of this offset must occur for the desired architecture.
- The align field is used to specify the power of 2 alignment the architecture requires.

```
struct fat_arch
{
    cpu_type_t cputype;
    cpu_subtype_t cpusubtype;
    uint32_t offset;
    uint32_t size;
    uint32_t align;
};
```

fm-unipack

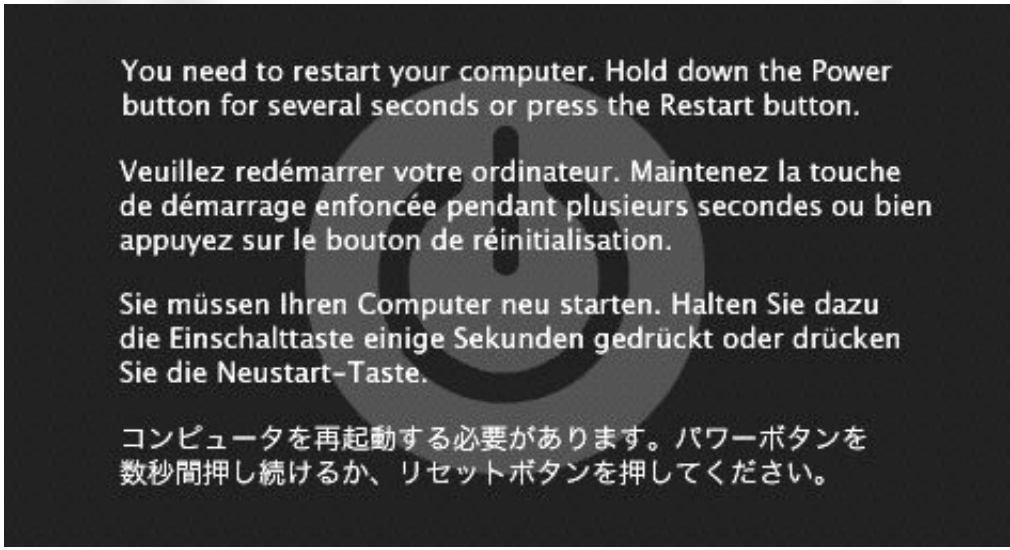
- Trivial tool I wrote for manipulating universal binaries.
- Demonstrates unpacking and packing a universal binary.
- Example.



SURESEC
SECURING THE SOURCE

Kernel Panics

- Many of my ideas for binary infection were cut short due to kernel panics.
- Maybe this is the immunity people mention? ;-)
- During research for this talk I triggered around 8 unique kernel panics.



suresec
SECURING THE SOURCE

Anti-Debugging Techniques

- OS X implements a *ptrace()* command called "***PTRACE_DENY_ATTACH***". When this is used the program will exit when an attempt to *ptrace()* it is made.
- Many people have difficulty parsing the mach-o headers correctly. Due to this many bugs exist in all of the common debuggers and dissemblers. (And also the Darwin Kernel ;-))

Anti-debugging techniques.. cont

- An example of one of these bugs is shown below.
- If you set the “number of sections” field in a SEGMENT_COMMAND to 0xffffffff many of the popular debuggers will crash. This bug exists in gdb (gnu debugger), IDA pro and the HTE editor.
- Amazingly this bug doesn’t exist in the Darwin kernel, therefore the binary executes correctly.

Conclusion

- Hopefully now you can see that Mac OS X, like all other operating systems, is exposed in exactly the same way to file system viruses.
- Thank you for listening to my talk.

suresec
SECURING THE SOURCE

Quotes

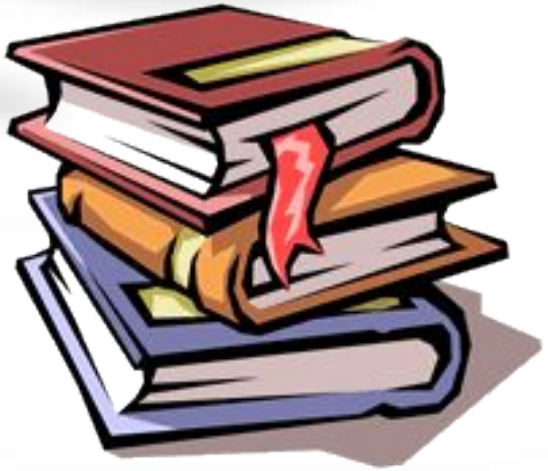
- "I am not and never was sold on "webtv" for a lot of reasons, (primarily having to do with how I personally use the Internet), but was not aware that it is immune to virus infestation. I though Apple was the only one who could make that claim. Learn something new every day."
- - http://forums.backpage.com/archive/index.php/t-194.html

- "Is this Mac running Mac OS X ?
- OK, you can stop worrying, it is not spyware. Mac OS X since release to present is totaly immune to virus/trojan/worm/spyware/adware/malware all these things are windows and PC things and Mac OS X users live absolutely free of any of that c**p.
- Thinking of getting a Mac now?
- Jim."
- - http://www.cybertechhelp.com/forums/showthread.php?t=66507

- "we believe that Apple has a solid operating system that has been to this point relatively immune to virus attacks."
- http://www.businessweek.com/bwdaily/dnflash/apr2005/nf20050415_4005_PG2.htm

References

- *http://en.wikipedia.org/wiki/Object_code*
- *http://en.wikipedia.org/wiki/Computer_virus*
- *http://en.wikipedia.org/wiki/Mach-O*
- *http://developer.apple.com/documentation/DeveloperTools/Conceptual/MachORuntime/MachORuntime.pdf*
- *http://developer.apple.com/documentation/MacOSX/Conceptual/universal_binary/*
- *http://www.l0t3k.org/biblio/magazine/english/b4b0/0009/b4b0-09.txt*
- *http://braden.machacking.net/bundle.html*



References

- http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/RuntimeOverview/chapter_4_section_1.html
- http://www.cocoadev.com/index.pl?ClassPosing