Let your Mach-O fly

Vincenzo Iozzo
snagg@sikurezza.org

Black Hat Briefings

Who am I?

- Student at Politecnico di Milano.
- Security Consultant at Secure Network srl.
- Reverse Engineer at Zynamics GmbH.

Goal of the talk

- In-memory execution of arbitrary binaries on a Mac OS X machine.

Talk outline

- Mach-O file structure
- XNU binary execution
- Attack technique
- Defeat ASLR on libraries to enhance the attack

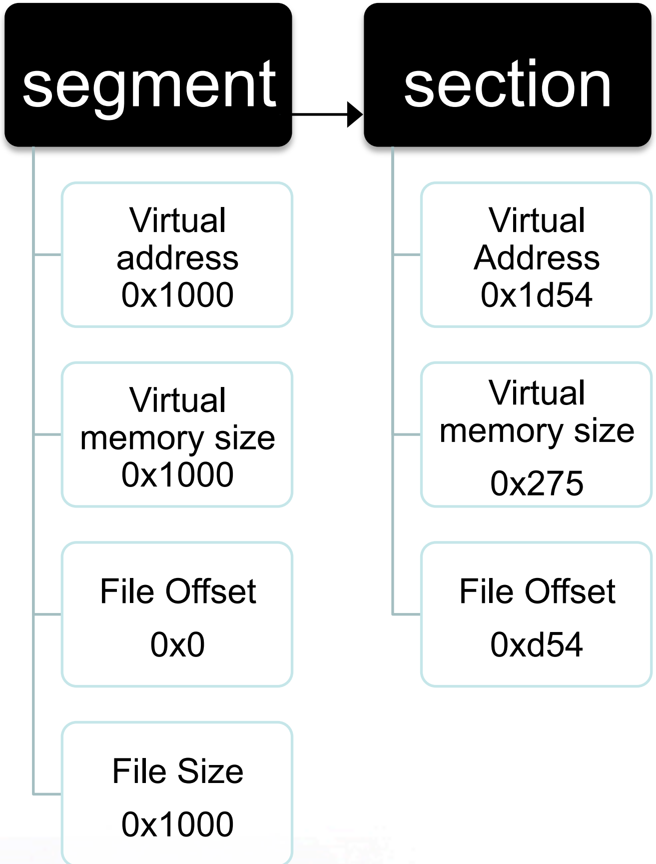
Talk outline

- **Mach-O file structure**
- XNU binary execution
- Attack technique
- Defeat ASLR on libraries to enhance the attack

Mach-O file

- **Header structure**: information on the target architecture and options to interpret the file.
- **Load commands**: symbol table location, registers state.
- **Segments**: define region of the virtual memory, contain sections with code or data.

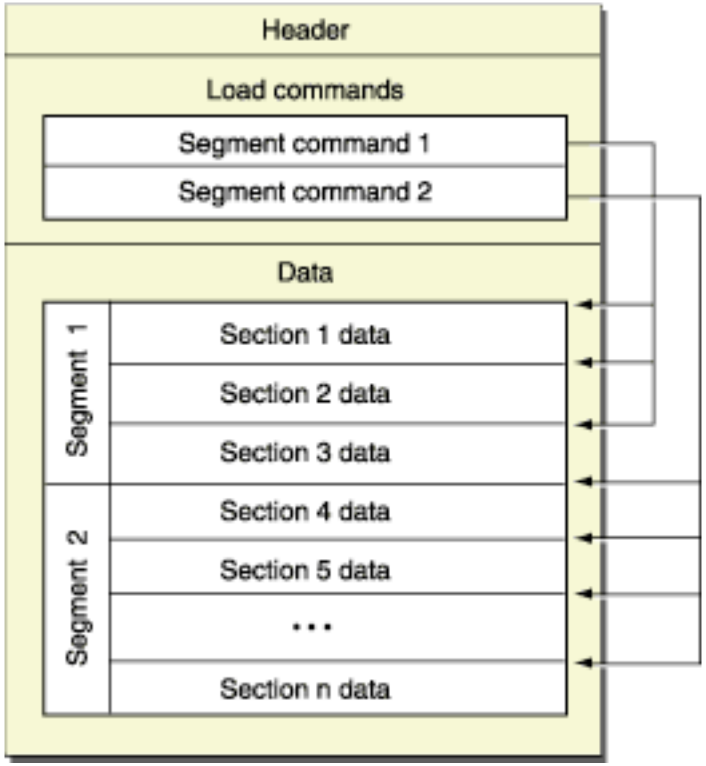
Segment and Sections



Important segments

- **__PAGEZERO**, if a piece of code accesses NULL it lands here. no protection flags.
- **__TEXT**, holds code and read-only data. RX protection.
- **__DATA**, holds data. RW protection.
- **__LINKEDIT**, holds information for the dynamic linker including symbol and string tables. RW protection.

Mach-O representation



Talk outline

- Mach-O file structure
- **XNU binary execution**
- Attack technique
- Defeat ASLR on libraries to enhance the attack

Binary execution

- Conducted by the kernel and the dynamic linker.
- The kernel, when finishes his part, jumps to the dynamic linker entry point.
- The dynamic linker is not randomized.

Execution steps

Kernel

- Maps the dynamic linker in the process address space.
- Parses the header structure and loads all segments.
- Creates a new stack.

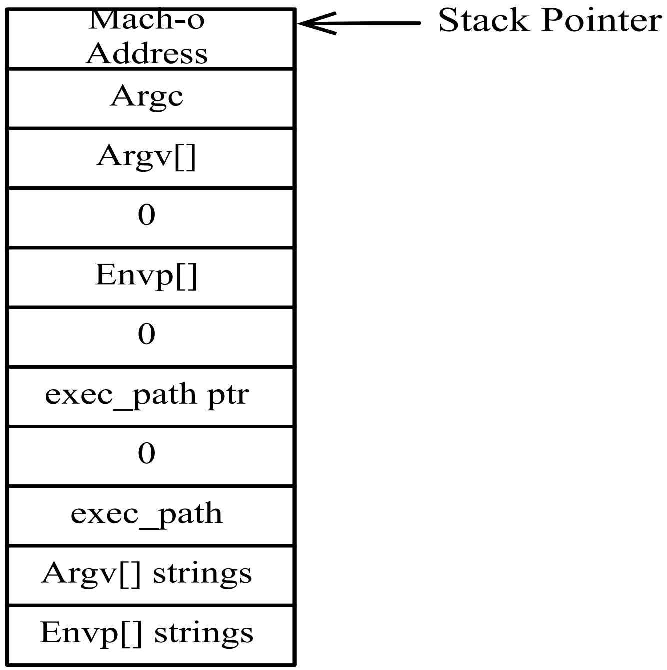
Dynamic linker

- Retrieves base address of the binary.
- Resolves symbols.
- Resolves library dependencies.
- Jumps to the binary entry point.

Stack

- Mach-O file base address.
- Command line arguments.
- Environment variables.
- Execution path.
- All padded.

Stack representation



Talk outline

- Mach-O file structure
- XNU binary execution
- **Attack technique**
- Defeat ASLR on libraries to enhance the attack

Proposed attack

- Userland-exec attack.
- Encapsulate a shellcode, aka auto-loader, and a crafted stack in the injected binary.
- Execute the auto-loader in the address space of the attacked process.

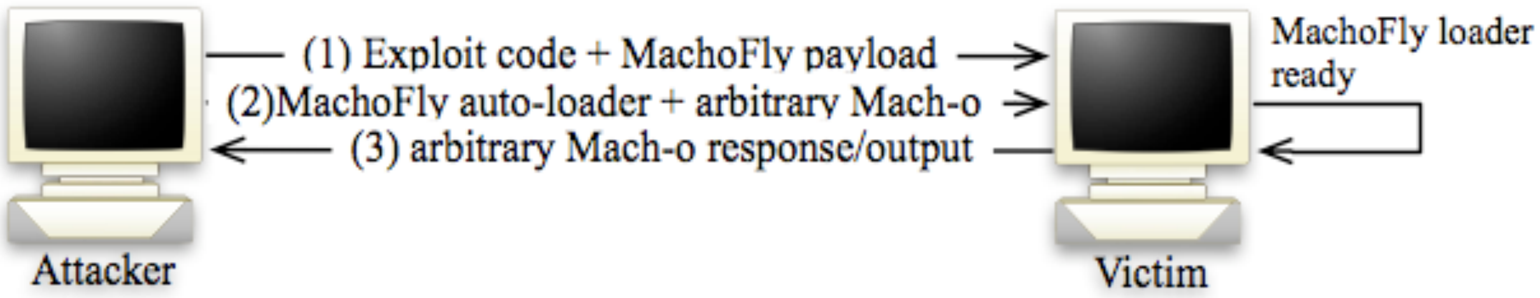
WWW

- Who: an attacker with a remote code execution in his pocket.
- Where: the attack is two-staged. First run a shellcode to receive the binary, then run the auto-loader contained in the binary.
- Why: later in this talk.

What kind of binaries?

Any Mach-O file, from ls to Safari

A nice picture



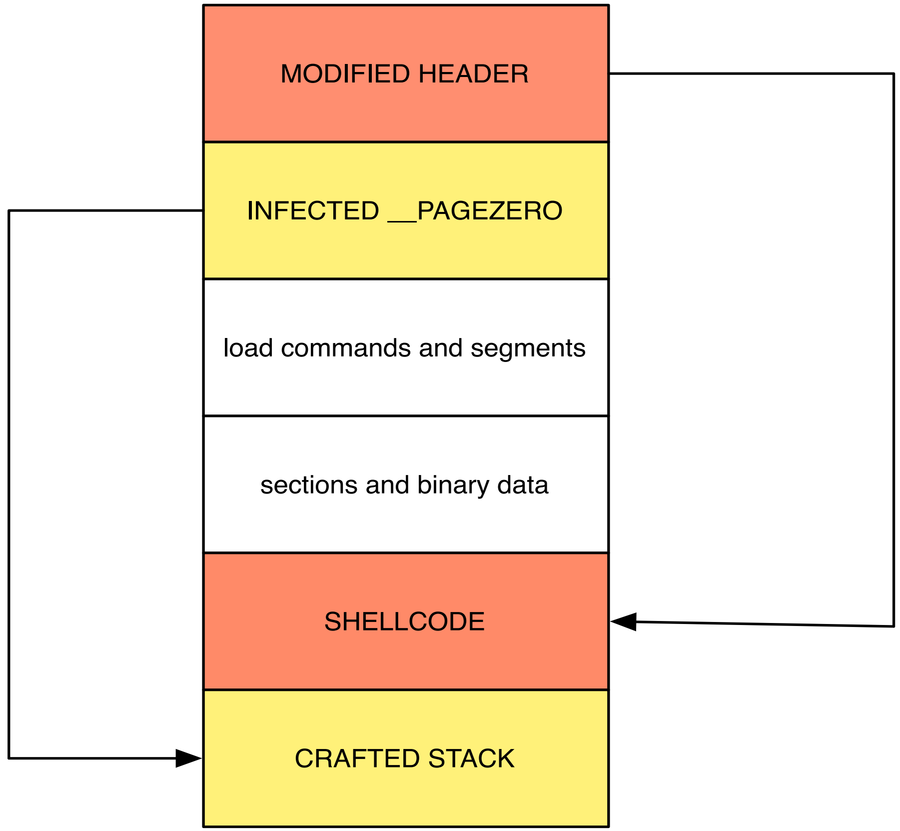
Infected binary

- We need to find a place to store the auto-loader and the crafted stack.
- __PAGEZERO infection technique.
- Cavity infector technique.

__PAGEZERO INFECTION

- Change __PAGEZERO protection flags with a custom value.
- Store the crafted stack and the auto-loader code at the end of the binary.
- Point __PAGEZERO to the crafted stack.
- Overwrite the first bytes of the file with the auto-loader address.

Binary layout



Auto-loader

- Impersonates the kernel.
- Un-maps the old binary.
- Maps the new one.

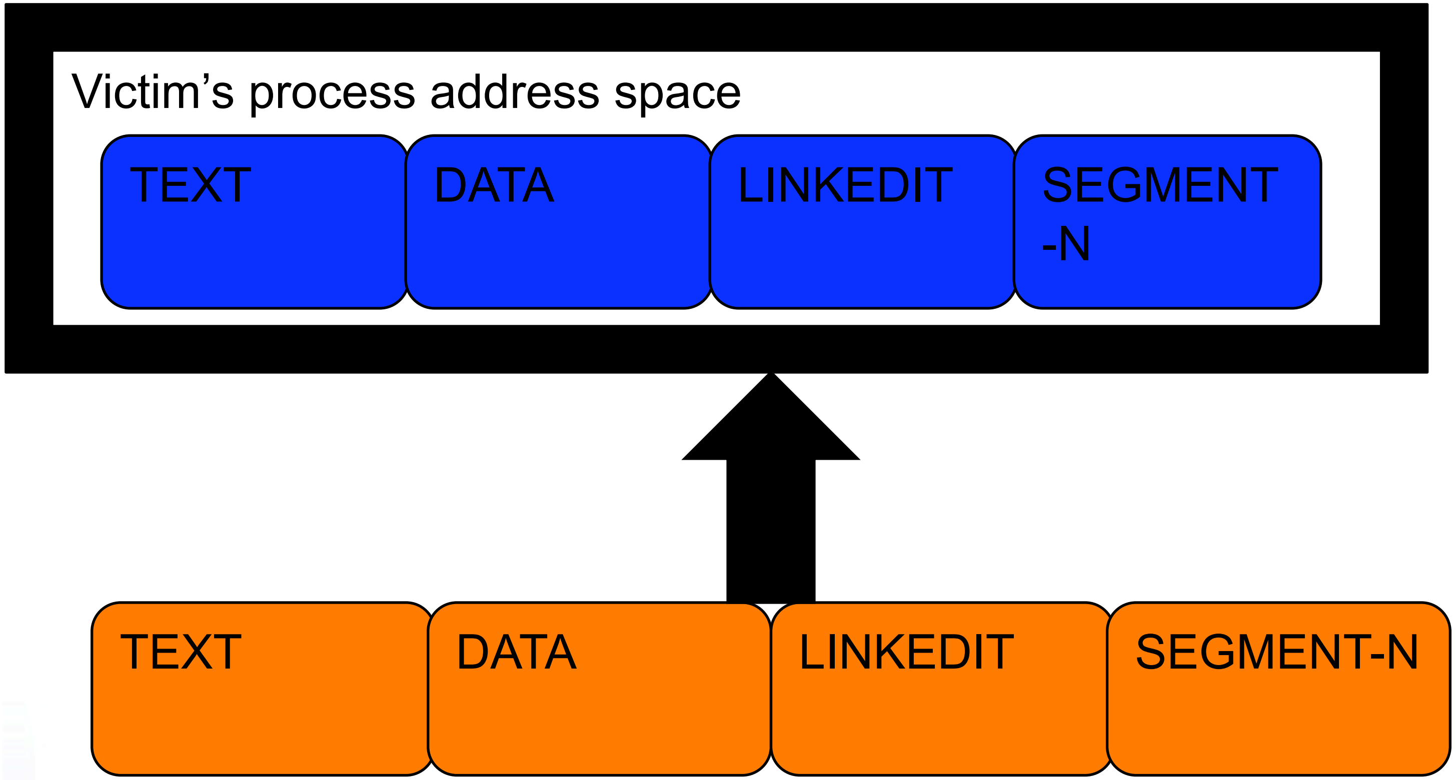
Auto-loader description

- Parses the binary.
- Reads the virtual addresses of the injected binary segments.
- Unloads the attacked binary segments pointed by the virtual addresses.
- Loads the injected binary segments.

Auto-loader description(2)

- Maps the crafted stack referenced by __PAGEZERO.
- Cleans registers.
- Cleans some libSystem variables.
- Jumps to dynamic linker entry point.

We do like pictures, don't we?



libSystem variables

- _malloc_def_zone_state
- _NXArgv_pointer
- _malloc_num_zones
- __keymgr_global

Why are those variables important?

- They are used in the initialization of malloc.
- Two of them are used for command line arguments parsing.
- Not cleaning them will result in a crash.

Hunts the variables

- Mac OS X Leopard has ASLR for libraries.
- Those variables are not exported.
- Cannot use dlopen()/dlsym() combo.

Talk outline

- Mach-O file structure
- XNU binary execution
- Attack technique
- **Defeat ASLR on libraries to enhance the attack**

Defeat ASLR

- Retrieve libSystem in-memory base address.
- Read symbols from the libSystem binary.
- Adjust symbols to the new address.

How ASLR works in Leopard

- Only libraries are randomized.
- The randomization is performed whenever the system or the libraries are updated.
- Library segments addresses are saved in dyld_shared_cache_arch.map.

Retrieve libSystem address

- Parse dyld_shared_cache_i386.map and search for libSystem entry.
- Adopt functions exported by the dynamic linker and perform the whole task in-memory.

Dyld functions

- **_dyld_image_count()** used to retrieve the number of linked libraries of a process.
- **_dyld_get_image_header()** used to retrieve the base address of each library.
- **_dyld_get_image_name()** used to retrieve the name of a given library.

Find ‘em

- Parse dyld load commands.
- Retrieve __LINKEDIT address.
- Iterate dyld symbol table and search for the functions name in __LINKEDIT.

Back to libSystem

- Non-exported symbols are taken out from the symbol table when loaded.
- Open libSystem binary, find the variables in the symbol table.
- Adjust variables to the base address of the in-memory __DATA segment.

Put pieces together

- Iterate the header structure of libSystem in-memory and find the __DATA base address.
  - __DATA base address 0x2000
  - Symbol at 0x2054
  - In-memory __DATA base address 0x4000
  - Symbol in-memory at 0x4054

Results

- Run a binary into an arbitrary machine.
- No traces on the hard-disk.
- No execve(), the kernel doesn't know about us.
- It works with every binary.
- It is possible to write payloads in a high level language.

Demo description

- Run a simple piece of code which acts like a shellcode and retrieve the binary.
- Execute the attack with nmap and Safari.
- Show network dump.
- Show memory layout before and after the attack.

DEMO

Future developments

- Employ encryption to avoid NIDS detection.
- Using cavity infector technique.
- Port the code to iPhone to evade code signing protection (Catch you at BH Europe).

Thanks, questions?