Playing with OS X

How to start your Apple reverse engineering adventure

fG! - Secuinside 2012

Who Am I

- An Economist and MBA.
- Computer enthusiast for the past 30 years.
- Someone who worked at one of the world’s best ATM networks, the Portuguese Multibanco.
- A natural-born reverser and assembler of all kinds of things, not just bits & bytes.

Introduction

- This presentation main goal is to allow you to make an easier transition into OS X reverse engineering world.
- I assume you already have some RE experience in other platforms, Windows or Unix.
- Many details are either minimal or omitted!

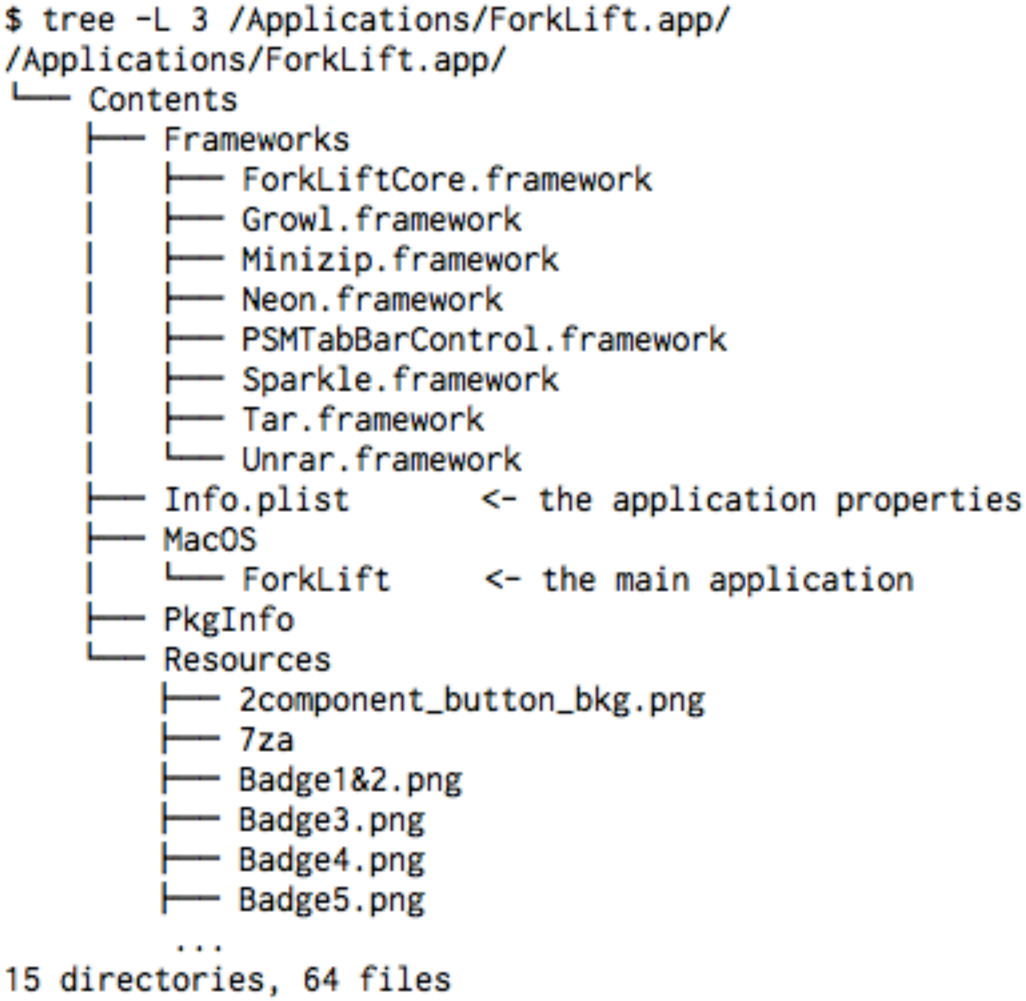
Summary

- Reversing in OS X - what’s different.
- Tools overview.
- Anatomy of a debugger.
- Anti-debugging.
- Code injection.
- Swizzling.
- Other tips & tricks.
- Reversing a crackme.
- Final remarks.

Reversing in OS X - what's different

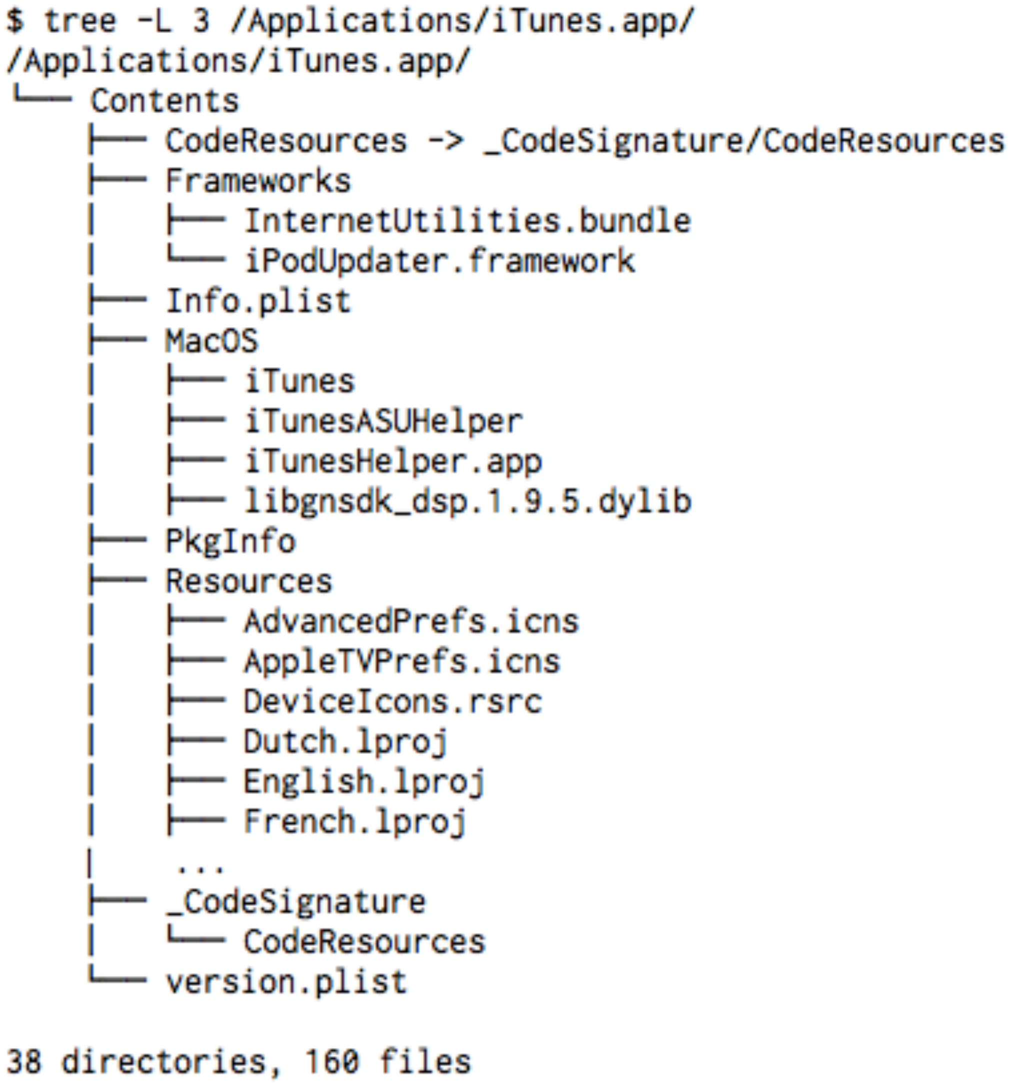
- Applications exist in bundle folders.
- These contain the application binary and other resources, such as:
  - Frameworks.
  - Language files.
  - Graphics, sounds, etc.
  - Code signatures, if applicable.
  - Application properties file, Info.plist.

Reversing in OS X - what’s different



```
$ tree -L 3 /Applications/ForkLift.app/
/Applications/ForkLift.app/
└── Contents
    ├── Frameworks
    │   ├── ForkLiftCore.framework
    │   ├── Growl.framework
    │   ├── Minizip.framework
    │   ├── Neon.framework
    │   ├── PSMTabBarControl.framework
    │   ├── Sparkle.framework
    │   ├── Tar.framework
    │   └── Unrar.framework
    ├── Info.plist       <- the application properties
    ├── MacOS
    │   └── ForkLift     <- the main application
    ├── PkgInfo
    └── Resources
        ├── 2component_button_bkg.png
        ├── 7za
        ├── Badge1&2.png
        ├── Badge3.png
        ├── Badge4.png
        ├── Badge5.png
        ...
15 directories, 64 files
```

Reversing in OS X - what’s different

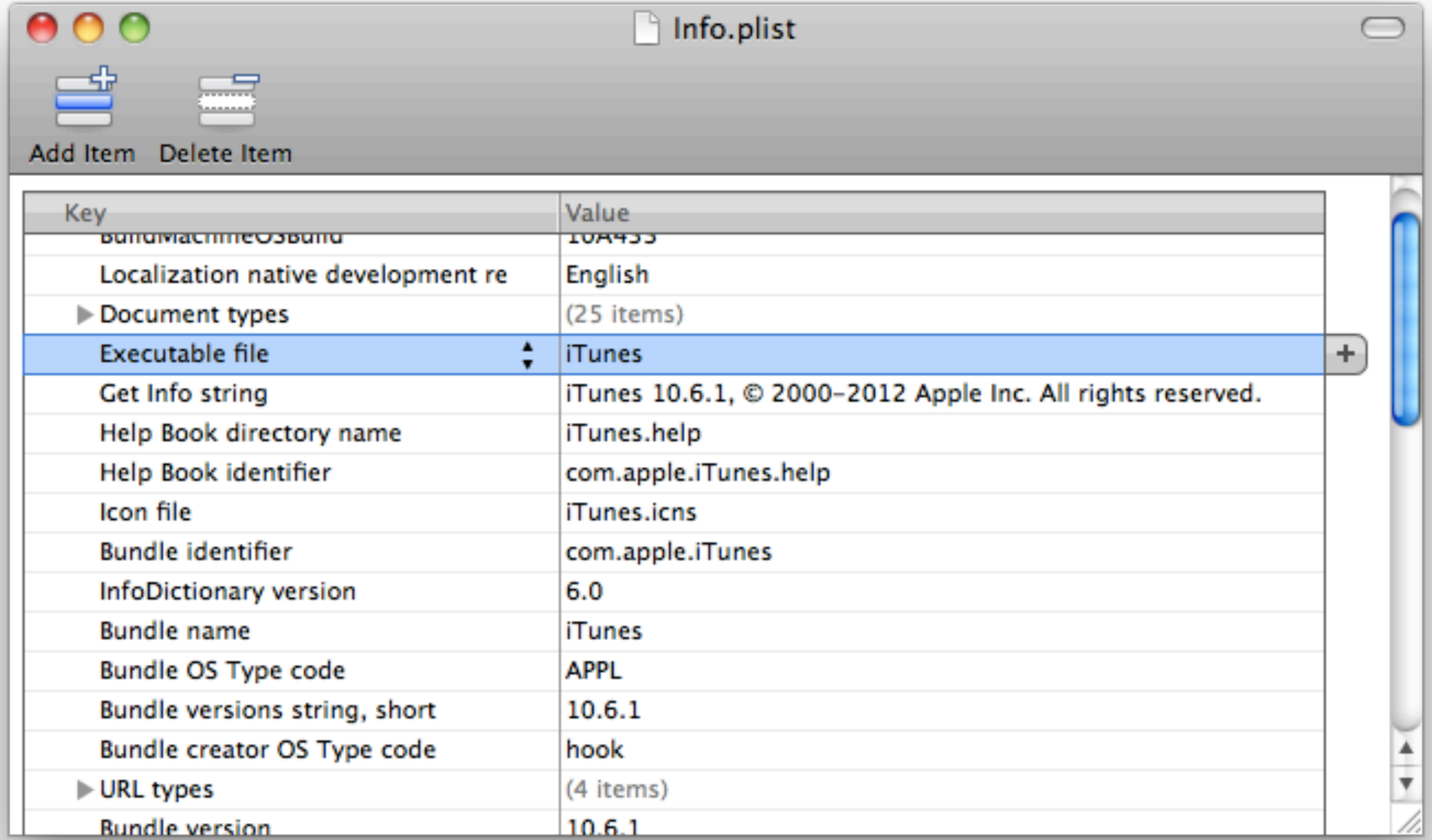


```
$ tree -L 3 /Applications/iTunes.app/
/Applications/iTunes.app/
└── Contents
    ├── CodeResources -> _CodeSignature/CodeResources
    ├── Frameworks
    │   ├── InternetUtilities.bundle
    │   └── iPodUpdater.framework
    ├── Info.plist
    ├── MacOS
    │   ├── iTunes
    │   ├── iTunesASUHelper
    │   ├── iTunesHelper.app
    │   ├── libgnsdk_dsp.1.9.5.dylib
    ├── PkgInfo
    ├── Resources
    │   ├── AdvancedPrefs.icns
    │   ├── AppleTVPrefs.icns
    │   ├── DeviceIcons.rsrc
    │   ├── Dutch.lproj
    │   ├── English.lproj
    │   ├── French.lproj
    |    ...
    ├── _CodeSignature
    │   └── CodeResources
    └── version.plist

38 directories, 160 files
```

Reversing in OS X - what’s different

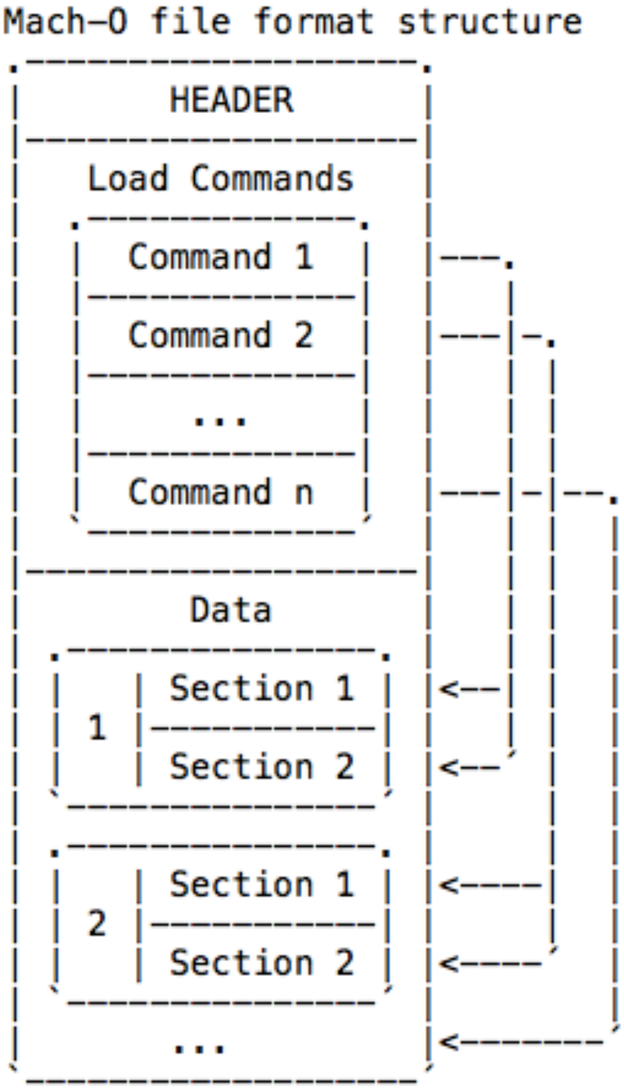
- The Info.plist contains useful information about the target application.
- For example, the CFBundleExecutable key gives you the name of the main executable.
- MacOS folder can contain more than one binary.
- I use it to collect some statistics about Mach-O binaries and also to find which binary to infect in my PoC virus.

Reversing in OS X - what’s different



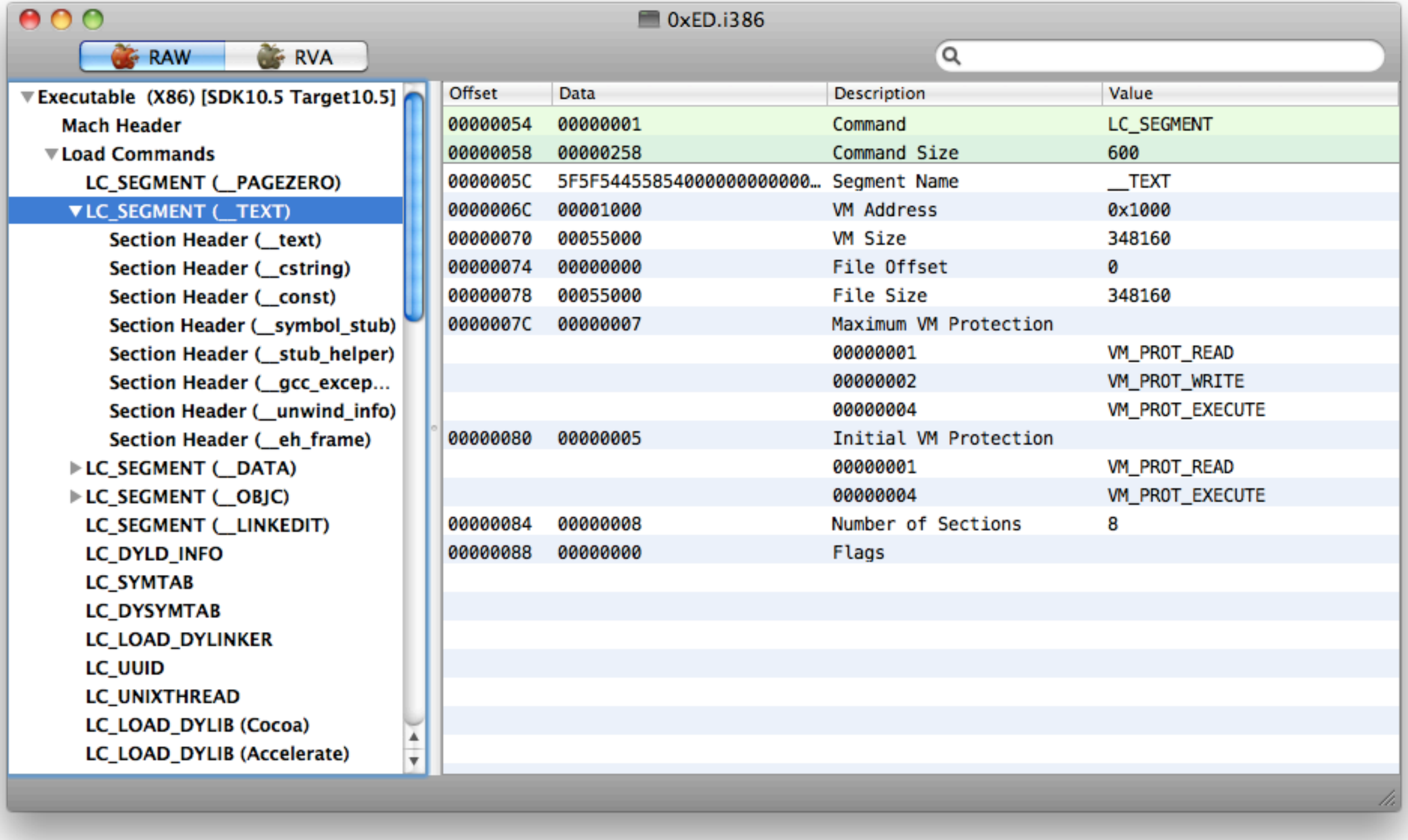
Key	Value
BuildMachineOSBuild	10A433
Localization native development re	English
▶ Document types	(25 items)
Executable file	iTunes
Get Info string	iTunes 10.6.1, © 2000–2012 Apple Inc. All rights reserved.
Help Book directory name	iTunes.help
Help Book identifier	com.apple.iTunes.help
Icon file	iTunes.icns
Bundle identifier	com.apple.iTunes
InfoDictionary version	6.0
Bundle name	iTunes
Bundle OS Type code	APPL
Bundle versions string, short	10.6.1
Bundle creator OS Type code	hook
▶ URL types	(4 items)
Bundle version	10.6.1

Reversing in OS X - what’s different



- Mach-O file format.
- Very simple!
- One header, with magic values 0xFEEDFACE (32bits) and 0xFEEDFACF (64bits).
- Followed by load commands and sections.
- And then data.

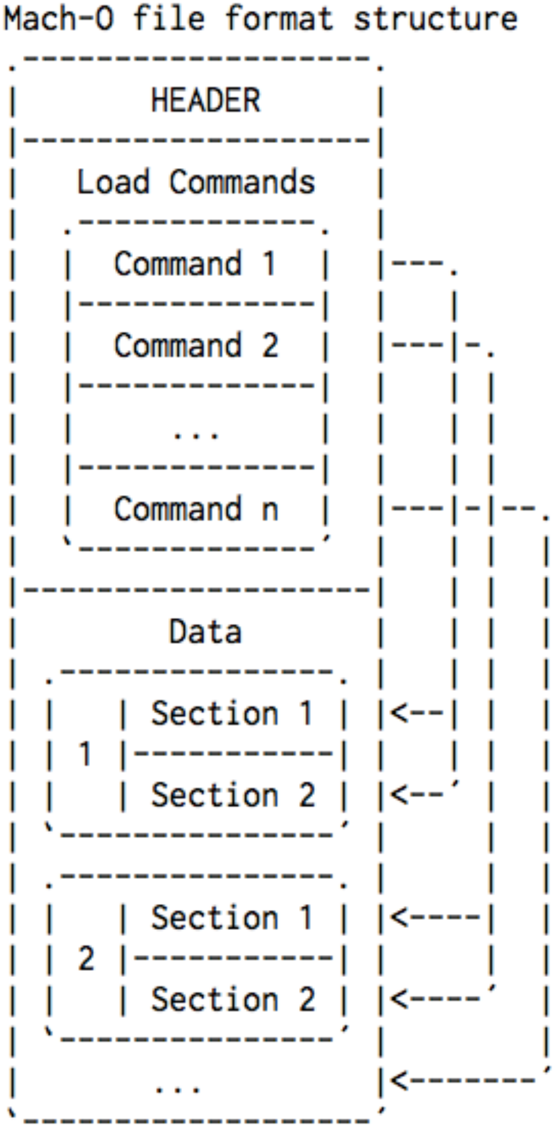
Reversing in OS X - what's different



- Executable (X86) [SDK10.5 Target10.5]
  - Mach Header
  - Load Commands
    - LC_SEGMENT (__PAGEZERO)
    - LC_SEGMENT (__TEXT)
      - Section Header (__text)
      - Section Header (__cstring)
      - Section Header (__const)
      - Section Header (__symbol_stub)
      - Section Header (__stub_helper)
      - Section Header (__gcc_excep...
      - Section Header (__unwind_info)
      - Section Header (__eh_frame)
    - LC_SEGMENT (__DATA)
    - LC_SEGMENT (__OBJC)
    - LC_SEGMENT (__LINKEDIT)
    - LC_DYLD_INFO
    - LC_SYMTAB
    - LC_DYSYMTAB
    - LC_LOAD_DYLINKER
    - LC_UUID
    - LC_UNIXTHREAD
    - LC_LOAD_DYLIB (Cocoa)
    - LC_LOAD_DYLIB (Accelerate)

Offset	Data	Description	Value
00000054	00000001	Command	LC_SEGMENT
00000058	00000258	Command Size	600
0000005C	5F5F544558540000000000000...	Segment Name	__TEXT
0000006C	00001000	VM Address	0x1000
00000070	00055000	VM Size	348160
00000074	00000000	File Offset	0
00000078	00055000	File Size	348160
0000007C	00000007	Maximum VM Protection	
		00000001	VM_PROT_READ
		00000002	VM_PROT_WRITE
		00000004	VM_PROT_EXECUTE
00000080	00000005	Initial VM Protection	
		00000001	VM_PROT_READ
		00000004	VM_PROT_EXECUTE
00000084	00000008	Number of Sections	8
00000088	00000000	Flags	

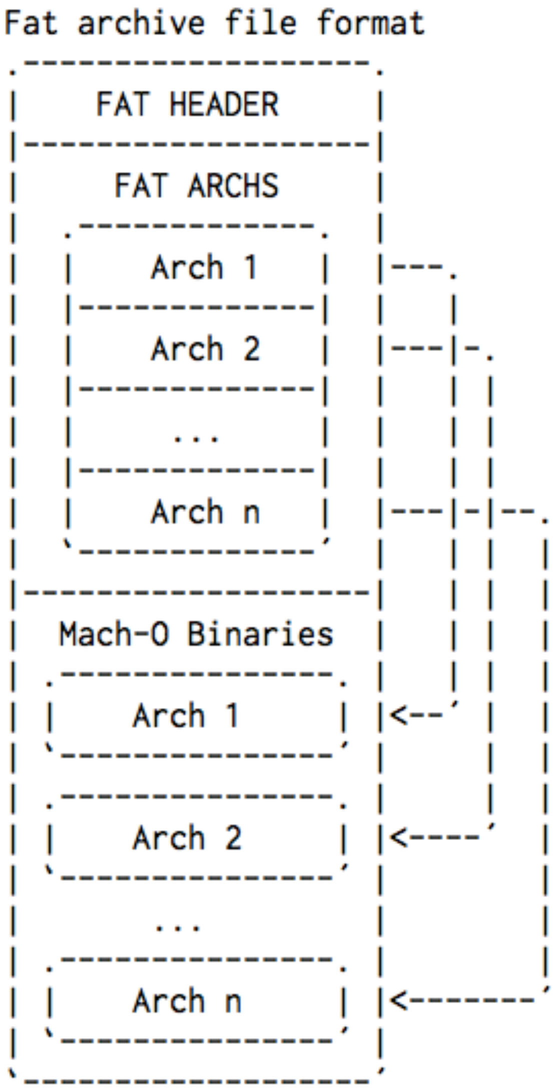
Reversing in OS X - what's different



```
Mach-O file format structure
.--------------------.
|       HEADER       |
|--------------------|
|   Load Commands    |
|  .--------------.  |
|  | Command 1    |  |---.
|  |--------------|  |   |
|  | Command 2    |  |---|-.
|  |--------------|  |   | |
|  |     ...      |  |   | |
|  |--------------|  |   | |
|  | Command n    |  |---|-|--.
|  `--------------'  |   | |  |
|--------------------|   | |  |
|        Data        |   | |  |
| .---------------.  |   | |  |
| |   | Section 1 | |<--| |  |
| | 1 |-----------| |   | |  |
| |   | Section 2 | |<--' |  |
| `---------------'  |     |  |
| .---------------.  |     |  |
| |   | Section 1 | |<----|  |
| | 2 |-----------| |     |  |
| |   | Section 2 | |<----'  |
| `---------------'  |        |
|                    |        |
|       ...          |<-------'
`--------------------'
```

- Code is located in __TEXT segment and __text section.
- Linked libraries in LC_LOAD_DYLIB commands.
- The entrypoint is defined at LC_UNIXTHREAD or LC_THREAD.
- Structs described at /usr/include/mach-o/loader.h.

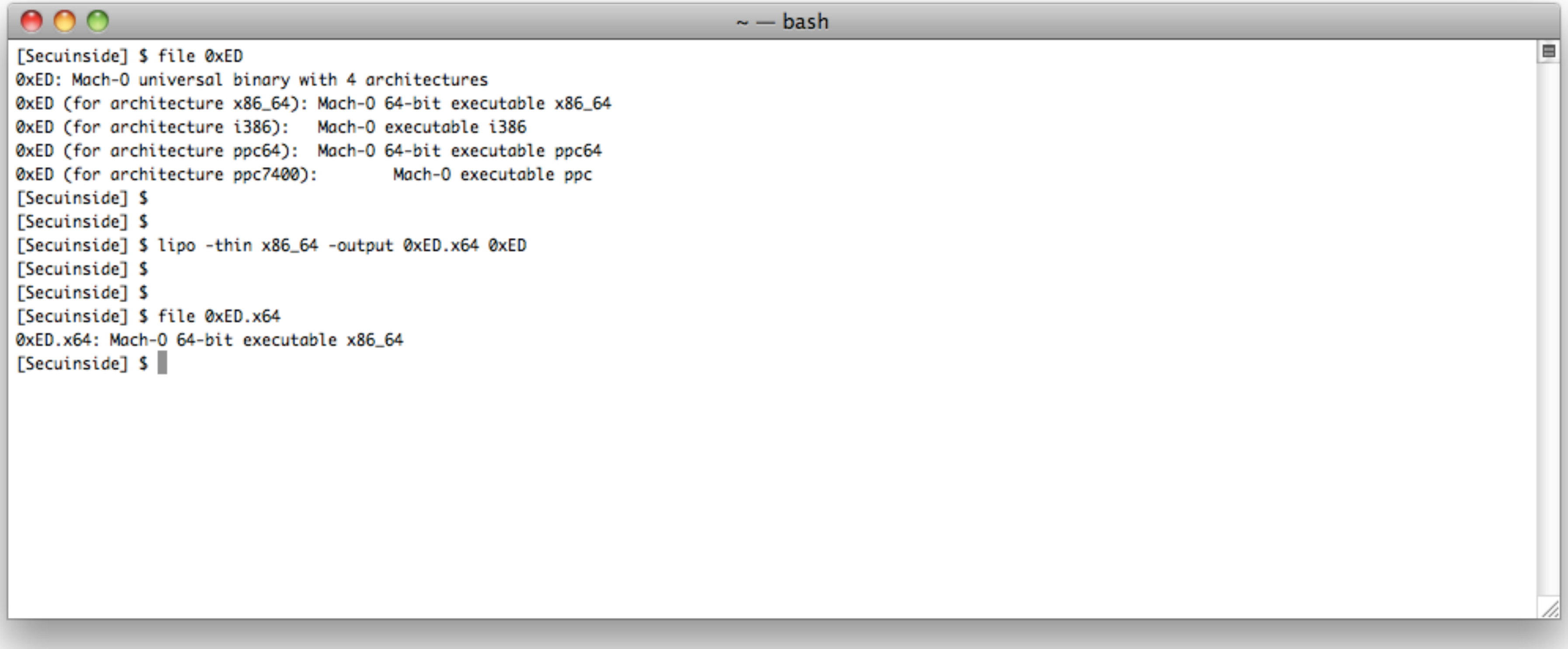
Reversing in OS X - what's different



```
Fat archive file format
.--------------------.
|     FAT HEADER     |
|--------------------|
|      FAT ARCHS     |
|  .-------------.   |
|  |    Arch 1   |   |---.
|  |-------------|   |   |
|  |    Arch 2   |   |---|-.
|  |-------------|   |   | |
|  |     ...     |   |   | |
|  |-------------|   |   | |
|  |    Arch n   |   |---|-|--.
|  '-------------'   |   | |  |
|--------------------|   | |  |
|   Mach-O Binaries  |   | |  |
| .--------------.   |   | |  |
| |    Arch 1    |   |<--' |  |
| '--------------'   |     |  |
| .--------------.   |     |  |
| |    Arch 2    |   |<----'  |
| '--------------'   |        |
|        ...         |        |
| .--------------.   |        |
| |    Arch n    |   |<-------'
| '--------------'   |
'--------------------'
```

- Fat archive:
- Allows to store different architectures inside a single "binary".
- Magic value is 0xCAFEBABE.
- Fat archive related structures are always big-endian!
- The "lipo" command allows you to extract a specific arch.

Reversing in OS X - what's different



```
[Secuinside] $ file 0xED
0xED: Mach-O universal binary with 4 architectures
0xED (for architecture x86_64): Mach-O 64-bit executable x86_64
0xED (for architecture i386):   Mach-O executable i386
0xED (for architecture ppc64):  Mach-O 64-bit executable ppc64
0xED (for architecture ppc7400):        Mach-O executable ppc
[Secuinside] $
[Secuinside] $
[Secuinside] $ lipo -thin x86_64 -output 0xED.x64 0xED
[Secuinside] $
[Secuinside] $
[Secuinside] $ file 0xED.x64
0xED.x64: Mach-O 64-bit executable x86_64
[Secuinside] $
```

Syntax:

lipo –thin [architecture] –output [output_file_name] fat_archive

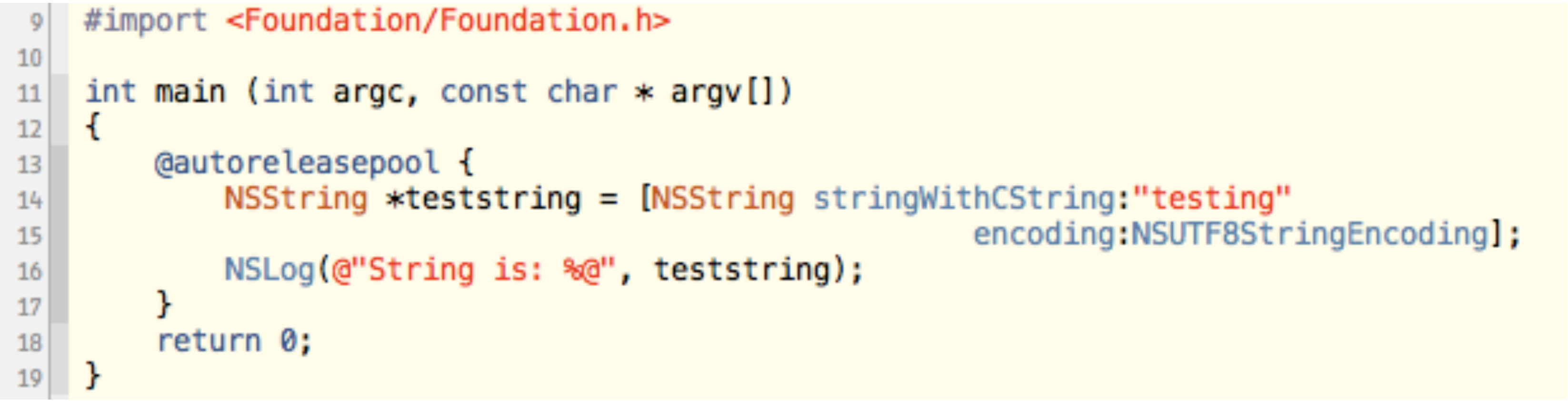
Reversing in OS X - what's different

- Objective-C.
- An extension to C language that enables objects to be created and manipulated.
- Rich set of frameworks: Cocoa, Cocoa Touch(iOS).
- Syntax of methods:
- [object message:arguments]
- [object message]

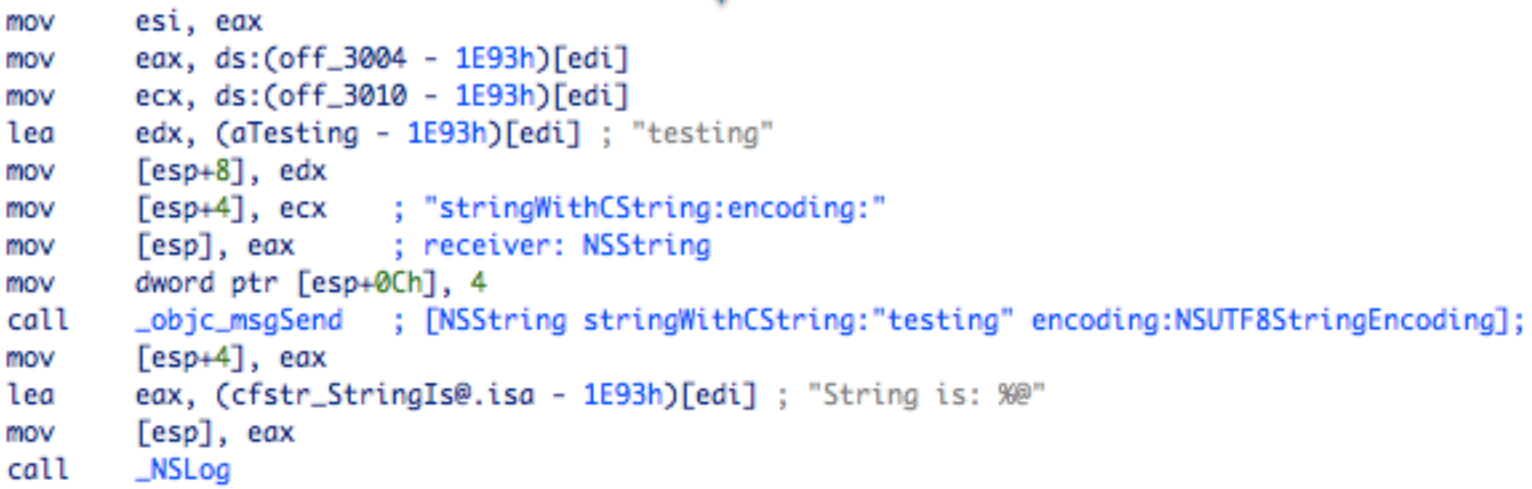
Reversing in OS X - what's different

- What happens on execution?
- There are no "traditional" calls to functions or methods.
- Instead, messages go thru the objc_msgSend function.
- id objc_msgSend(id theReceiver, SEL theSelector, ...)
- There are three more message functions, but objc_msgSend is the most common.
- Check Objective-C Runtime Reference documentation.
- Also nemo's article at Phrack #66.

Reversing in OS X - what’s different



```objc
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[])
{
    @autoreleasepool {
        NSString *teststring = [NSString stringWithCString:"testing"
                                                  encoding:NSUTF8StringEncoding];
        NSLog(@"String is: %@", teststring);
    }
    return 0;
}
```



```
mov     esi, eax
mov     eax, ds:(off_3004 - 1E93h)[edi]
mov     ecx, ds:(off_3010 - 1E93h)[edi]
lea     edx, (aTesting - 1E93h)[edi] ; "testing"
mov     [esp+8], edx
mov     [esp+4], ecx    ; "stringWithCString:encoding:"
mov     [esp], eax      ; receiver: NSString
mov     dword ptr [esp+0Ch], 4
call    _objc_msgSend   ; [NSString stringWithCString:"testing" encoding:NSUTF8StringEncoding];
mov     [esp+4], eax
lea     eax, (cfstr_StringIs@.isa - 1E93h)[edi] ; "String is: %@"
mov     [esp], eax
call    _NSLog
```

Reversing in OS X - what's different

- Those messages can be traced:
- With GDB.
- With DTrace.
- Nemo's article has sample code for the above solutions.
- The GDB version works great in iOS.
- Set NSObjCMessageLoggingEnabled environment variable to YES and messages will be logged to /tmp/msgSends-pid.
- More info at Technical Note TN2124 – Mac OS X Debugging Magic.

Tools overview

- Quality, quantity, and number of features of tools lags a lot versus the Windows world.
- Especially in GUI applications.
- This is slowly improving with increased interest in this platform.
- Download Apple’s command line tools for Xcode or the whole Xcode. (https://developer.apple.com/downloads/ , requires free Apple ID).

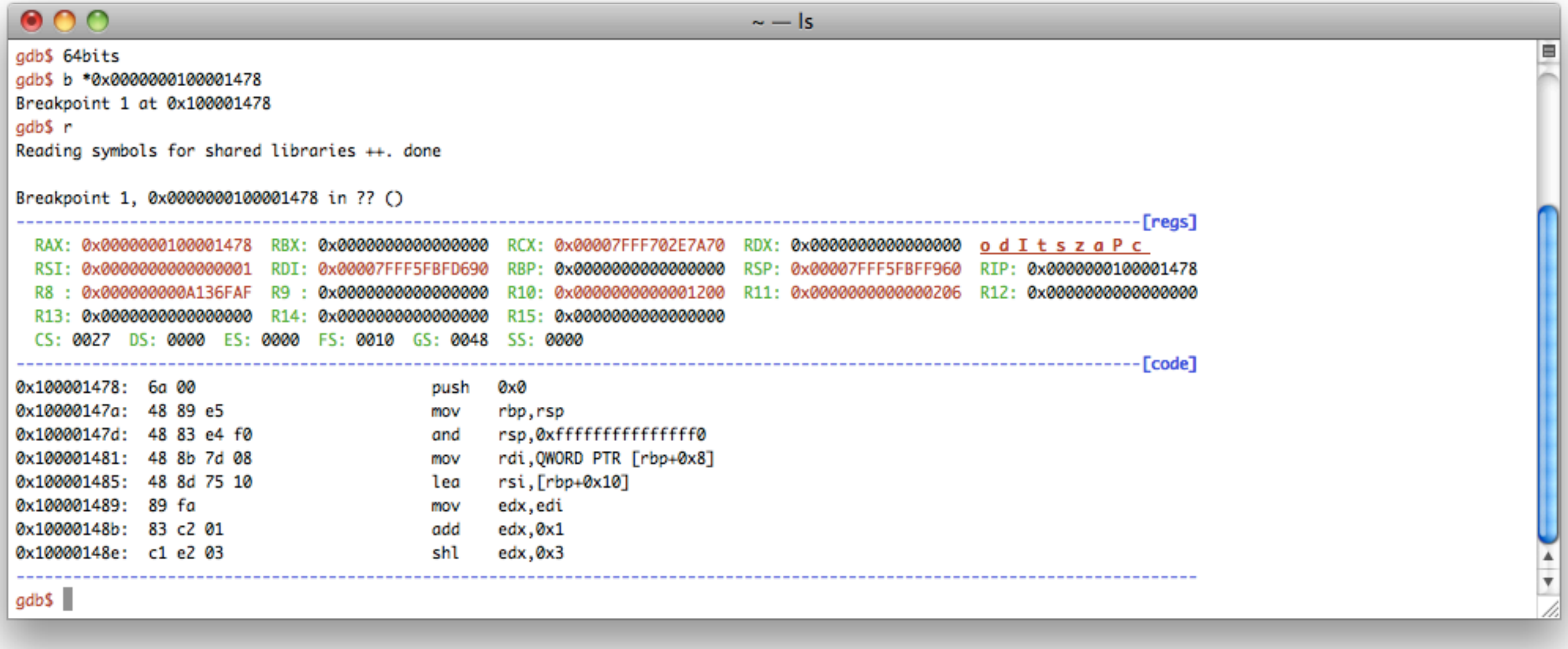
Tools overview - Debuggers

- GDB.
- IDA.
- PyDBG/PyDBG64.
- Radare.
- LLDB.
- Hopper.
- Forget about GNU GDB 7.x !

Tools overview - Debuggers

- GDB is my favourite.
- Apple forked it at 6.x - stopped in time.
- Lots of bugs, missing features - LLDB is the new thing.
- But, it does the job!
- Use my patches (http://reverse.put.as/patches/).
- And gdbinit, to have that retro Softice look & features (http://reverse.put.as/gdbinit/).
- Please read the header of gdbinit!

Tools overview - Debuggers



```
gdb$ 64bits
gdb$ b *0x0000000100001478
Breakpoint 1 at 0x100001478
gdb$ r
Reading symbols for shared libraries ++. done

Breakpoint 1, 0x0000000100001478 in ?? ()
-----------------------------------------------------------------------------------------------------------------------[regs]
  RAX: 0x0000000100001478  RBX: 0x0000000000000000  RCX: 0x00007FFF702E7A70  RDX: 0x0000000000000000  o d I t s z a P c 
  RSI: 0x0000000000000001  RDI: 0x00007FFF5FBFD690  RBP: 0x0000000000000000  RSP: 0x00007FFF5FBFF960  RIP: 0x0000000100001478
  R8 : 0x000000000A136FAF  R9 : 0x0000000000000000  R10: 0x0000000000001200  R11: 0x0000000000000206  R12: 0x0000000000000000
  R13: 0x0000000000000000  R14: 0x0000000000000000  R15: 0x0000000000000000
  CS: 0027  DS: 0000  ES: 0000  FS: 0010  GS: 0048  SS: 0000
-----------------------------------------------------------------------------------------------------------------------[code]
0x100001478:  6a 00                         push   0x0
0x10000147a:  48 89 e5                      mov    rbp,rsp
0x10000147d:  48 83 e4 f0                   and    rsp,0xfffffffffffffff0
0x100001481:  48 8b 7d 08                   mov    rdi,QWORD PTR [rbp+0x8]
0x100001485:  48 8d 75 10                   lea    rsi,[rbp+0x10]
0x100001489:  89 fa                         mov    edx,edi
0x10000148b:  83 c2 01                      add    edx,0x1
0x10000148e:  c1 e2 03                      shl    edx,0x3
-----------------------------------------------------------------------------------------------------------------------------
gdb$ 
```

Tools overview – GDB commands

- Add software breakpoints with “b, tb, bp, bpt”.
- Add hardware breakpoints with “hb, thb, bhb, bht”.
- To breakpoint on memory location you must add the * before address. Example: b *0x1000.
- Step thru code with “next(n), nexti(ni), step, stepi”.
- Step over calls with “stepo, stepoh”.
- Change flags register with “cf*” commands.
- Evaluate and print memory with “x” and “print”.

Tools overview – GDB commands

- Print Object-C objects with “po”.
- Modify memory with “set”.
- Register: set $eax = 0x31337.
- Memory: set *(int*)0x1000 = 0x31337.
- Assemble instructions using “asm”.
- Dump memory with dump commands (“dump memory” is probably the one you will use often).
- Find about all gdbinit commands with “help user”.

Tools overview - Disassemblers

- Otool, with –tV option. The objdump equivalent.
- OTX – enhanced otool output (AT&T syntax).
- IDA – native version so no more Windows VM.
- Hopper – the new kid on the block, actively developed, very cheap, includes a decompiler.
- Home-made disassembler using Distorm3 or any other disassembler library (udis86, libdasm also work well).

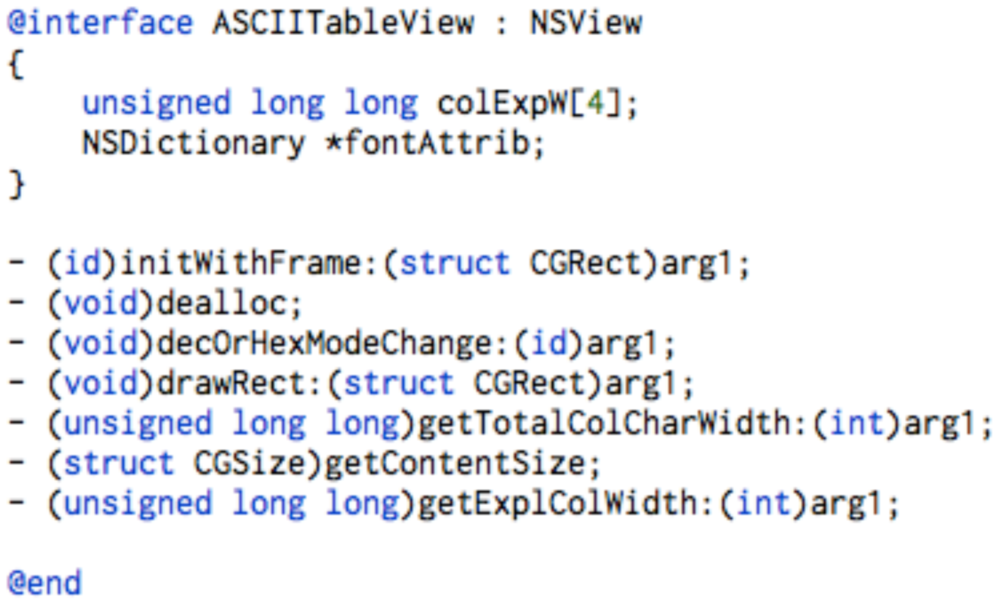
Tools overview – Other tools

- MachOView – great visual replacement for otool –l.
- Hex-editors: 0xED, Hex Fiend, 010 Editor, etc.
- nm – displays symbols list.
- vmmap – display virtual memory map of a process.
- DTrace. Check [9] for some useful scripts.
- File system usage: fs_usage.

Tools overview – Class-dump

- Allows you to examine the available Objective-C information.
- Generates the declarations for the classes, categories and protocols.
- Useful to understand the internals and design of Objective-C apps.
- Used a lot by the iOS jailbreak community.

Tools overview – Class-dump



```
@interface ASCIITableView : NSView
{
    unsigned long long colExpW[4];
    NSDictionary *fontAttrib;
}

- (id)initWithFrame:(struct CGRect)arg1;
- (void)dealloc;
- (void)decOrHexModeChange:(id)arg1;
- (void)drawRect:(struct CGRect)arg1;
- (unsigned long long)getTotalColCharWidth:(int)arg1;
- (struct CGSize)getContentSize;
- (unsigned long long)getExplColWidth:(int)arg1;

@end
```

Mach tasks and threads

- Explaining the whole Mac OS X architecture would require a whole presentation.
- Others did it before, please check [20] and [21].
- For now we just need one concept.
- Unix process abstraction is split into tasks and threads.
- Tasks contain the resources and do not execute code.
- Threads execute within a task and share its resources.
- A BSD process has a one-to-one mapping with a Mach task.

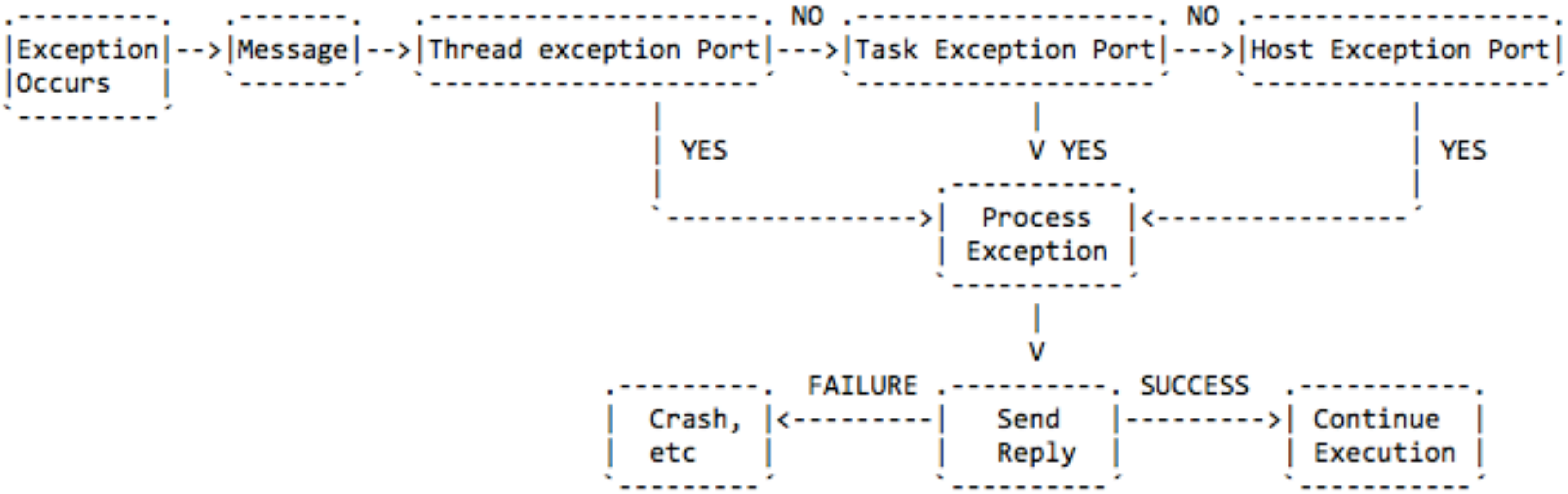
Anatomy of a debugger

- OS X ptrace implementation is incomplete (and useless).
- Mach exceptions are the solution.
- Each task has three levels of exception ports: thread, task, host.
- Exceptions are converted to messages and sent to those ports.
- Messages are received and processed by the exception handler.

Anatomy of a debugger

- The exception handler can be located in another task, usually a debugger.
- Or another thread in the same task.
- Kernel expects a reply message with success or failure.
- Messages are first delivered to the most specific port.
- Detailed information on Chapter 9.7 of Mac OS X Internals.

Anatomy of a debugger



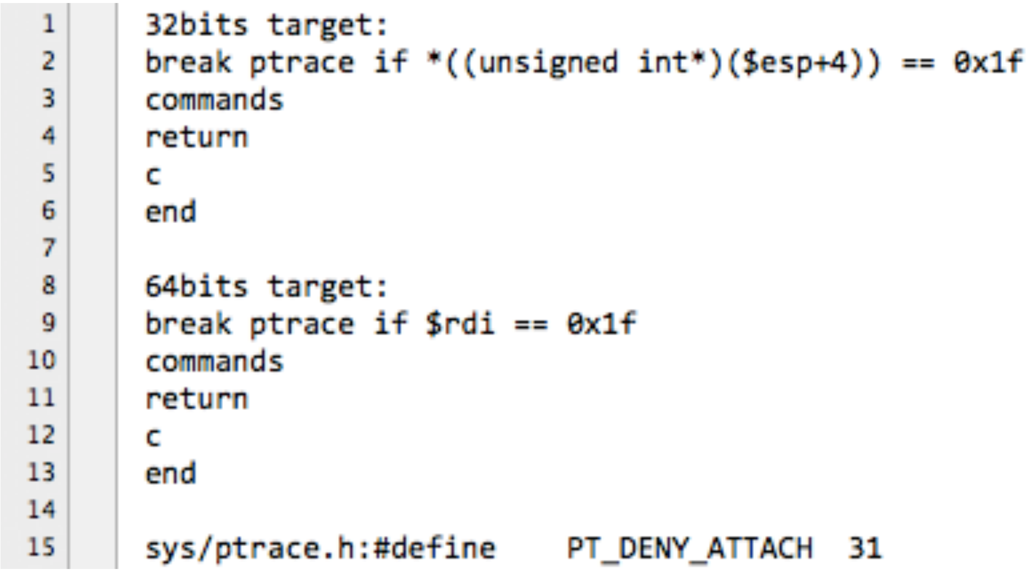
```
.----------.   .-------.   .----------------------. NO .-------------------. NO .-------------------.
|Exception|-->|Message|-->|Thread exception Port|--->|Task Exception Port|--->|Host Exception Port|
|Occurs    |   `-------'   `----------------------'    `-------------------'    `-------------------'
`----------'                        |                            |                          |
                                    | YES                        V YES                      | YES
                                    |                     .-----------.                     |
                                    `------------------>| Process    |<--------------------'
                                                          | Exception |
                                                          `-----------'
                                                                |
                                                                V
                     .---------.  FAILURE  .-----------.  SUCCESS  .-----------.
                     | Crash,  |<----------|   Send    |--------->| Continue  |
                     | etc     |           |   Reply   |          | Execution |
                     `---------'           `-----------'          `-----------'
```

Anatomy of a debugger

- By default, the thread exception ports are set to null and task exception ports are inherited during fork().
- We need access to the task port.
- Not a problem if debugging from the same task: mach_task_self().
- Higher privileges required (root or procmod group) if from another task: task_for_pid().

Anti-debugging – "Old school"

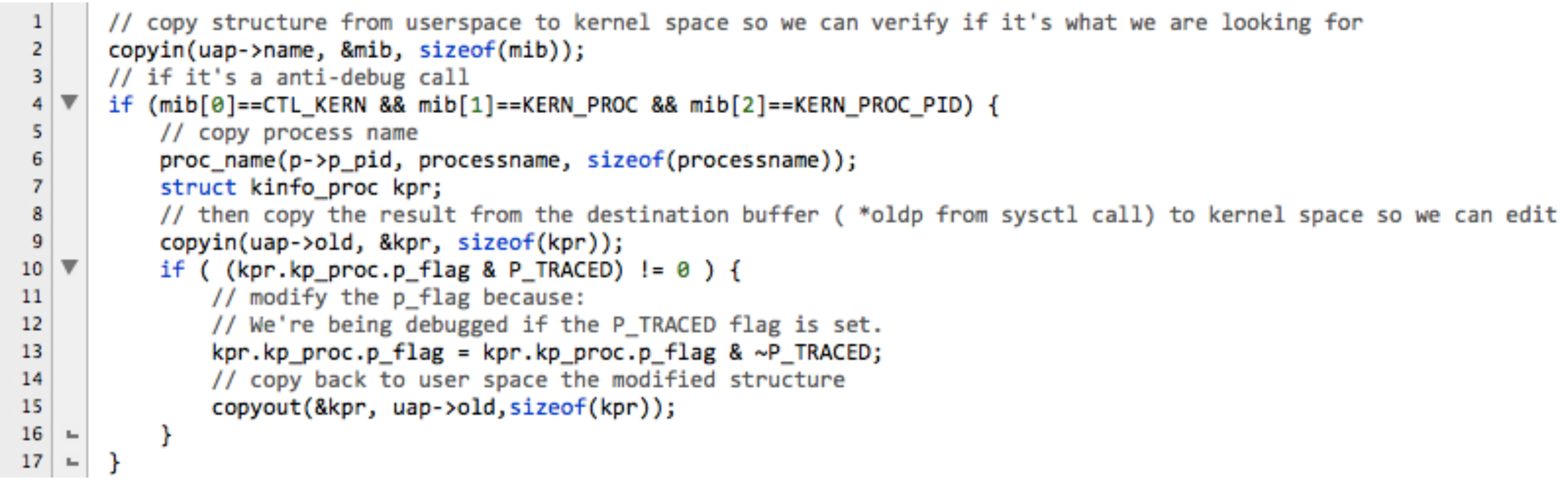
- ptrace(PT_DENY_ATTACH, …).
- Ok, that was a joke. This is useless!
- Just breakpoint on ptrace() or use a kernel module.



```
32bits target:
break ptrace if *((unsigned int*)($esp+4)) == 0x1f
commands
return
c
end

64bits target:
break ptrace if $rdi == 0x1f
commands
return
c
end

sys/ptrace.h:#define    PT_DENY_ATTACH  31
```

Anti-debugging – “Old school”

- AmIBeingDebugged() from Apple’s Technote QA1361.
- Calls sysctl() and verifies if P_TRACED flag is set in proc structure.
- Breakpoint sysctl() and modify the result or use a kernel module.

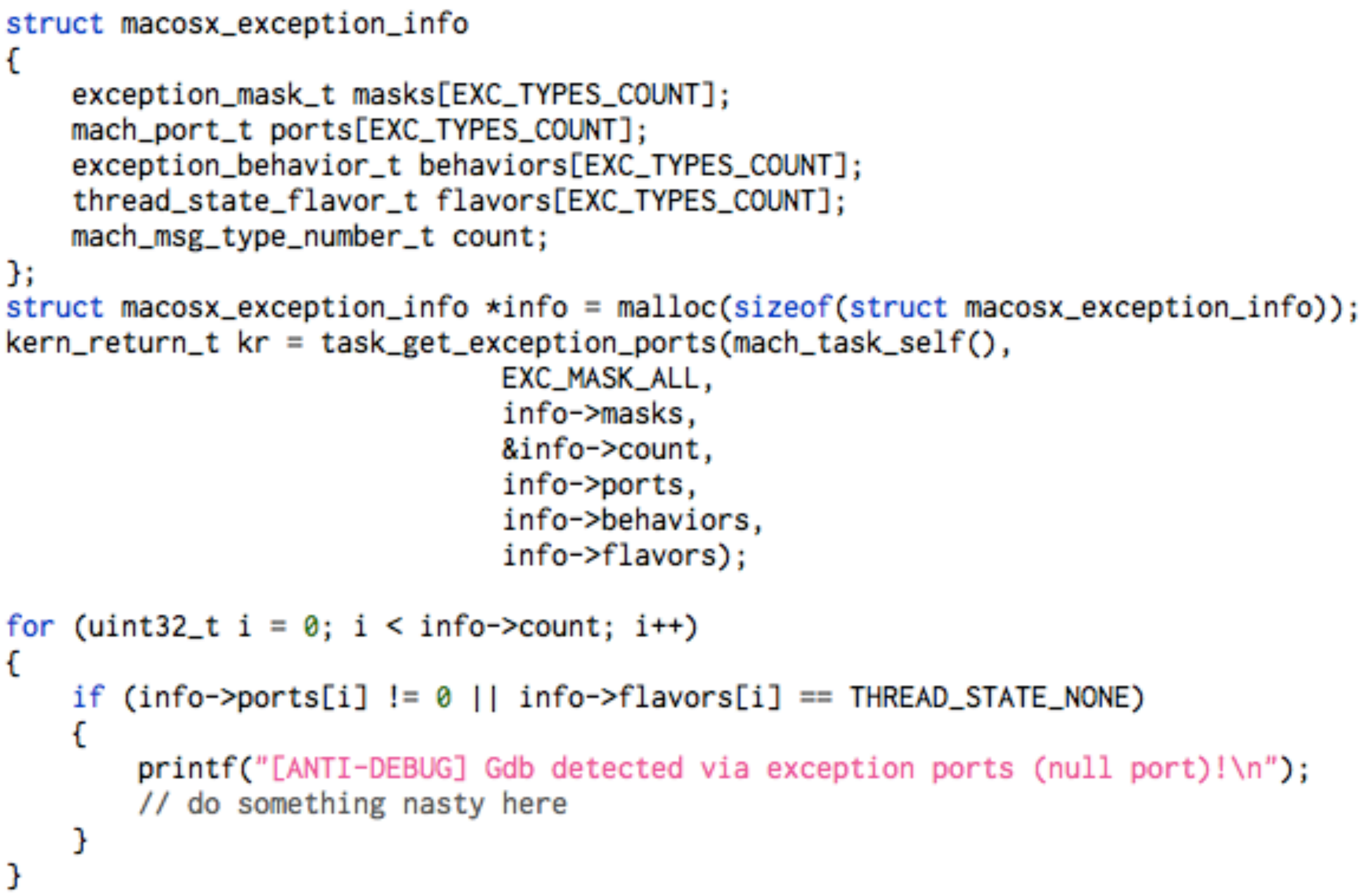


```
// copy structure from userspace to kernel space so we can verify if it's what we are looking for
copyin(uap->name, &mib, sizeof(mib));
// if it's a anti-debug call
if (mib[0]==CTL_KERN && mib[1]==KERN_PROC && mib[2]==KERN_PROC_PID) {
    // copy process name
    proc_name(p->p_pid, processname, sizeof(processname));
    struct kinfo_proc kpr;
    // then copy the result from the destination buffer ( *oldp from sysctl call) to kernel space so we can edit
    copyin(uap->old, &kpr, sizeof(kpr));
    if ( (kpr.kp_proc.p_flag & P_TRACED) != 0 ) {
        // modify the p_flag because:
        // We're being debugged if the P_TRACED flag is set.
        kpr.kp_proc.p_flag = kpr.kp_proc.p_flag & ~P_TRACED;
        // copy back to user space the modified structure
        copyout(&kpr, uap->old,sizeof(kpr));
    }
}
```

Anti-debugging - #1

- Remember, debuggers “listen” on the exception ports.
- We can verify if that port is set.
- Use task_get_exception_ports().
- GDB uses a mask of EXC_MASK_ALL and a flavour of THREAD_STATE_NONE.
- Iterate thru all the ports and verify if port is different than NULL.
- Do something (nasty) ☺.

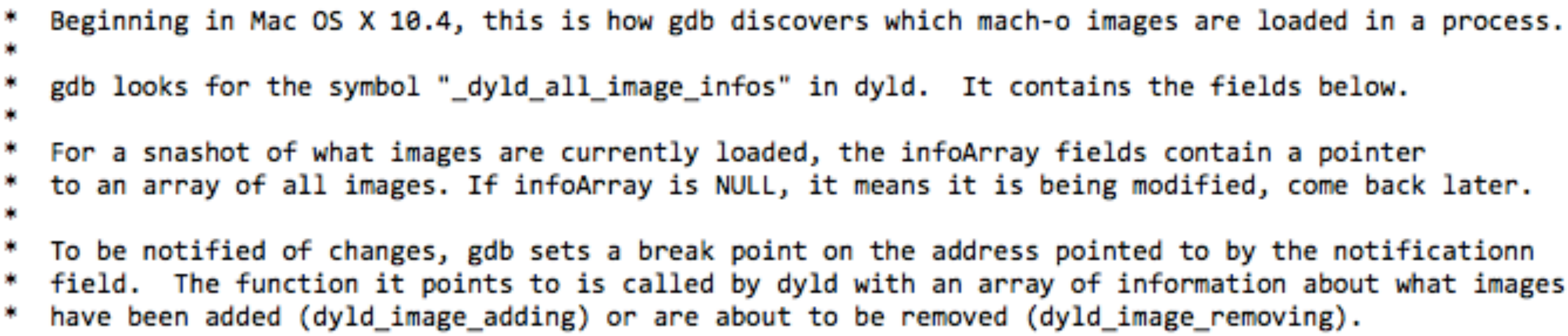
Anti-debugging - #1



```c
struct macosx_exception_info
{
    exception_mask_t masks[EXC_TYPES_COUNT];
    mach_port_t ports[EXC_TYPES_COUNT];
    exception_behavior_t behaviors[EXC_TYPES_COUNT];
    thread_state_flavor_t flavors[EXC_TYPES_COUNT];
    mach_msg_type_number_t count;
};
struct macosx_exception_info *info = malloc(sizeof(struct macosx_exception_info));
kern_return_t kr = task_get_exception_ports(mach_task_self(),
                              EXC_MASK_ALL,
                              info->masks,
                              &info->count,
                              info->ports,
                              info->behaviors,
                              info->flavors);

for (uint32_t i = 0; i < info->count; i++)
{
    if (info->ports[i] != 0 || info->flavors[i] == THREAD_STATE_NONE)
    {
        printf("[ANTI-DEBUG] Gdb detected via exception ports (null port)!\n");
        // do something nasty here
    }
}
```

Anti-debugging - #2

- Check for GDB breakpoint.
- GDB is notified by dyld when new images are added to the process.
- This is what allows the GDB “stop-on-solib-events” trick that I used to get into Pace’s protection.
- Symbol name is _dyld_all_image_info.

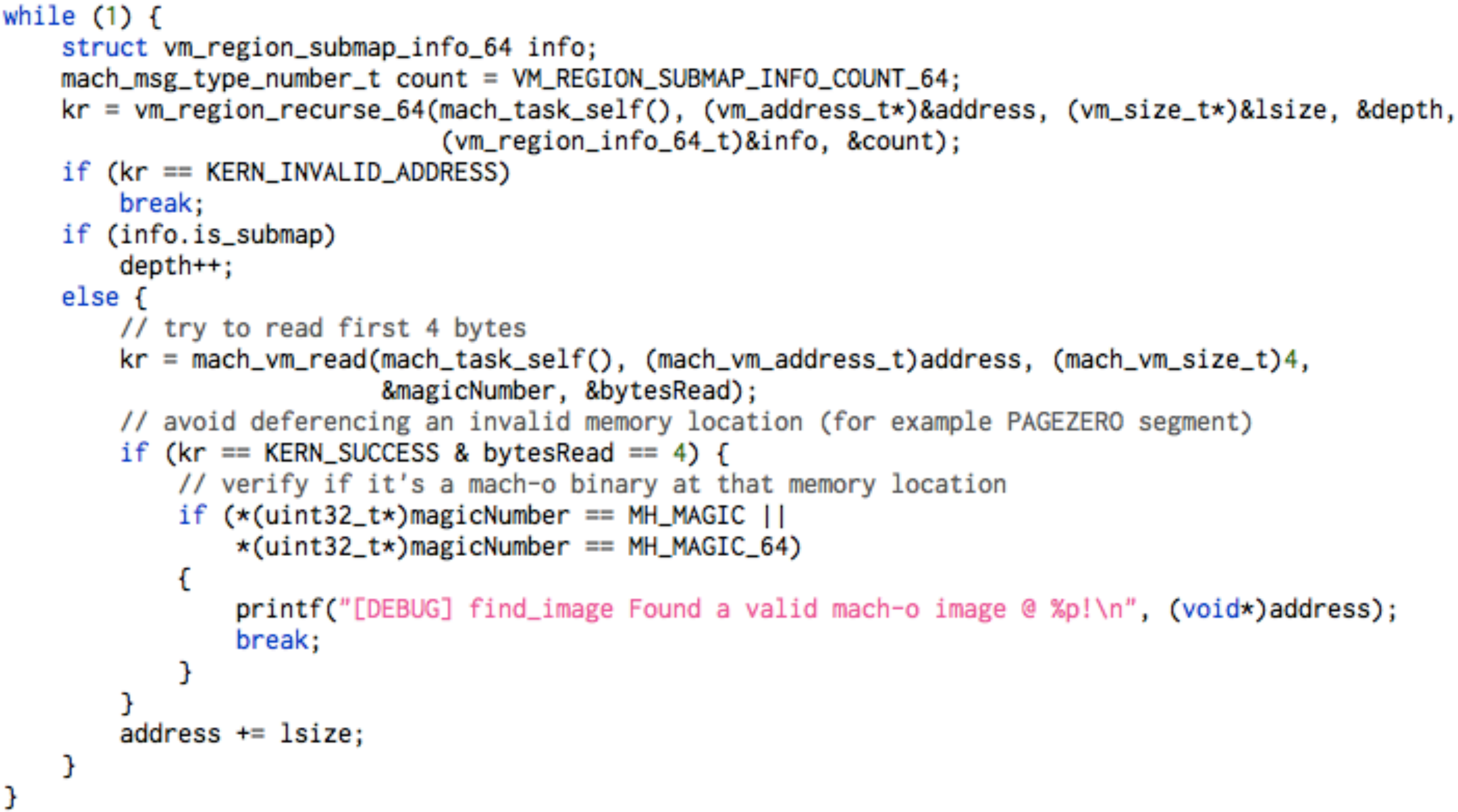


```
*   Beginning in Mac OS X 10.4, this is how gdb discovers which mach-o images are loaded in a process.
*
*   gdb looks for the symbol "_dyld_all_image_infos" in dyld.  It contains the fields below.
*
*   For a snashot of what images are currently loaded, the infoArray fields contain a pointer
*   to an array of all images. If infoArray is NULL, it means it is being modified, come back later.
*
*   To be notified of changes, gdb sets a break point on the address pointed to by the notificationn
*   field.  The function it points to is called by dyld with an array of information about what images
*   have been added (dyld_image_adding) or are about to be removed (dyld_image_removing).
```

Anti-debugging - #2

- How to do it:
- Use vm_region_recurse_64() to iterate thru memory.
- We need a starting point.
- Dyld stays at 0x8FExxxxx area in 32 bits processes.
- And at 0x00007FFFxxxxxxxx area in 64 bits processes.
- It's always the first image in that area, even with ASLR.
- Try to find a valid Mach-O image by searching for the magic value.

Anti-debugging - #2



```c
while (1) {
    struct vm_region_submap_info_64 info;
    mach_msg_type_number_t count = VM_REGION_SUBMAP_INFO_COUNT_64;
    kr = vm_region_recurse_64(mach_task_self(), (vm_address_t*)&address, (vm_size_t*)&lsize, &depth,
                              (vm_region_info_64_t)&info, &count);
    if (kr == KERN_INVALID_ADDRESS)
        break;
    if (info.is_submap)
        depth++;
    else {
        // try to read first 4 bytes
        kr = mach_vm_read(mach_task_self(), (mach_vm_address_t)address, (mach_vm_size_t)4,
                          &magicNumber, &bytesRead);
        // avoid deferencing an invalid memory location (for example PAGEZERO segment)
        if (kr == KERN_SUCCESS & bytesRead == 4) {
            // verify if it's a mach-o binary at that memory location
            if (*(uint32_t*)magicNumber == MH_MAGIC ||
                *(uint32_t*)magicNumber == MH_MAGIC_64)
            {
                printf("[DEBUG] find_image Found a valid mach-o image @ %p!\n", (void*)address);
                break;
            }
        }
        address += lsize;
    }
}
```

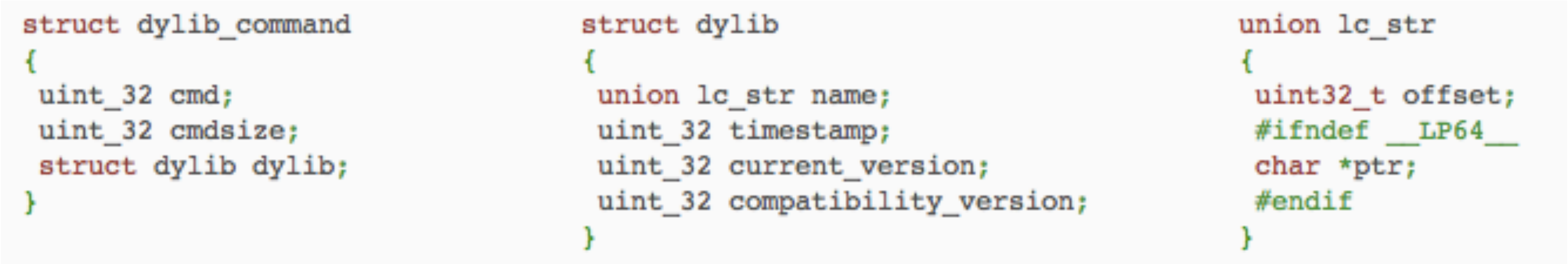
Anti-debugging - #2

- Add DYLD_ALL_IMAGE_INFOS_OFFSET_OFFSET to the base address of dyld image.
- Get a pointer to the dyld_all_image_infos structure.
- We are interested in the notification field.
- Verify if there’s a INT3 on that address.
- Do something (nasty) ☺.

Anti-debugging - #3

- This one crashes GDB on load, but not if attached.
- Abuse the specification of struct dylib_command.
- The library name is usually after the structure.
- And offset field points there.
- Just put the string somewhere else and modify the offset accordingly.
- Check http://reverse.put.as/2012/01/31/anti-debug-trick-1-abusing-mach-o-to-crash-gdb/.

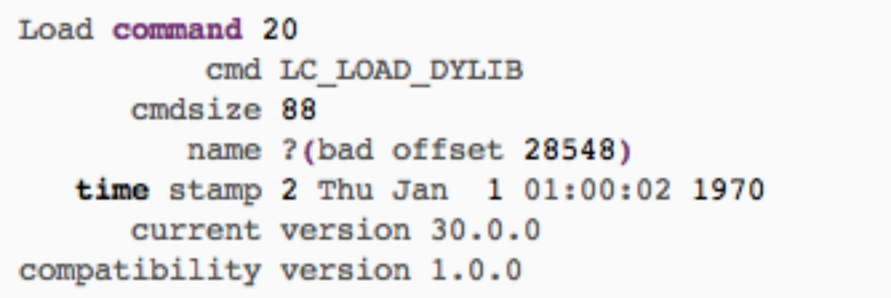
Anti-debugging - #3



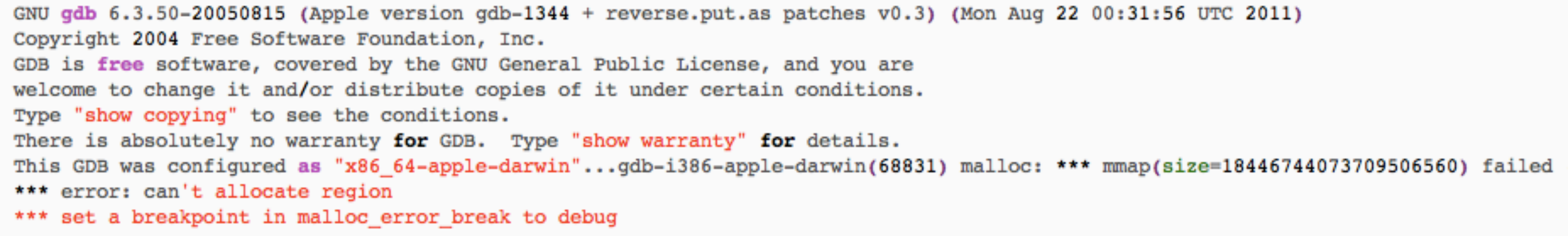
```
struct dylib_command
{
 uint_32 cmd;
 uint_32 cmdsize;
 struct dylib dylib;
}
```

```
struct dylib
{
 union lc_str name;
 uint_32 timestamp;
 uint_32 current_version;
 uint_32 compatibility_version;
}
```

```
union lc_str
{
 uint32_t offset;
 #ifndef __LP64__
 char *ptr;
 #endif
}
```



```
Load command 20
          cmd LC_LOAD_DYLIB
      cmdsize 88
         name ?(bad offset 28548)
   time stamp 2 Thu Jan  1 01:00:02 1970
      current version 30.0.0
compatibility version 1.0.0
```



```
GNU gdb 6.3.50-20050815 (Apple version gdb-1344 + reverse.put.as patches v0.3) (Mon Aug 22 00:31:56 UTC 2011)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "x86_64-apple-darwin"...gdb-i386-apple-darwin(68831) malloc: *** mmap(size=18446744073709506560) failed
*** error: can't allocate region
*** set a breakpoint in malloc_error_break to debug
```

Kernel debugging

- The default solution is to use two computers, via Ethernet or Firewire.
- VMware can be used, which is so much better.
- The traditional way, using Apple’s kernel debugger protocol with GDB.
- Or VMware’s built in debug server also with GDB.
- Check out my original post and snare’s updates at http://ho.ax.

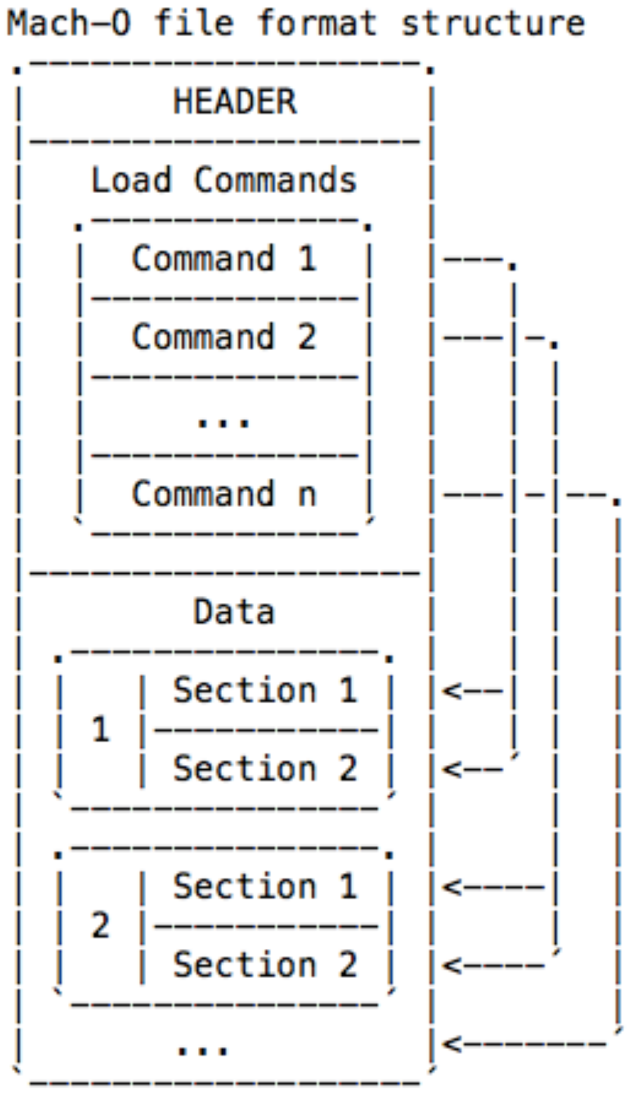
Code injection

- DYLD_INSERT_LIBRARIES is equivalent to LD_PRELOAD.
- I prefer another trick!
- Modify the Mach-O header and add a new command: LC_LOAD_DYLIB.
- Most binaries have enough space to do this.

Code injection

- What can it be used for?
- A run-time patcher.
- A debugger & tracer.
- A virus (the subject of my next presentation).
- Function hijacking & method swizzling.
- Anti-piracy & DRM.
- Something else!

Code injection



```
Mach-O file format structure
.-------------------.
|      HEADER       |
|-------------------|
|   Load Commands   |
|  .-------------.  |
|  |  Command 1  |  |---.
|  |-------------|  |   |
|  |  Command 2  |  |---|-.
|  |-------------|  |   | |
|  |     ...     |  |   | |
|  |-------------|  |   | |
|  |  Command n  |  |---|-|--.
|  `-------------'  |   | |  |
|-------------------|   | |  |
|       Data        |   | |  |
| .---------------. |   | |  |
| |   | Section 1 | |<--| |  |
| | 1 |-----------| |   | |  |
| |   | Section 2 | |<--' |  |
| `---------------' |     |  |
| .---------------. |     |  |
| |   | Section 1 | |<----|  |
| | 2 |-----------| |     |  |
| |   | Section 2 | |<----'  |
| `---------------' |        |
|        ...        |<-------'
`-------------------'
```

Some stats from my /Applications folder:

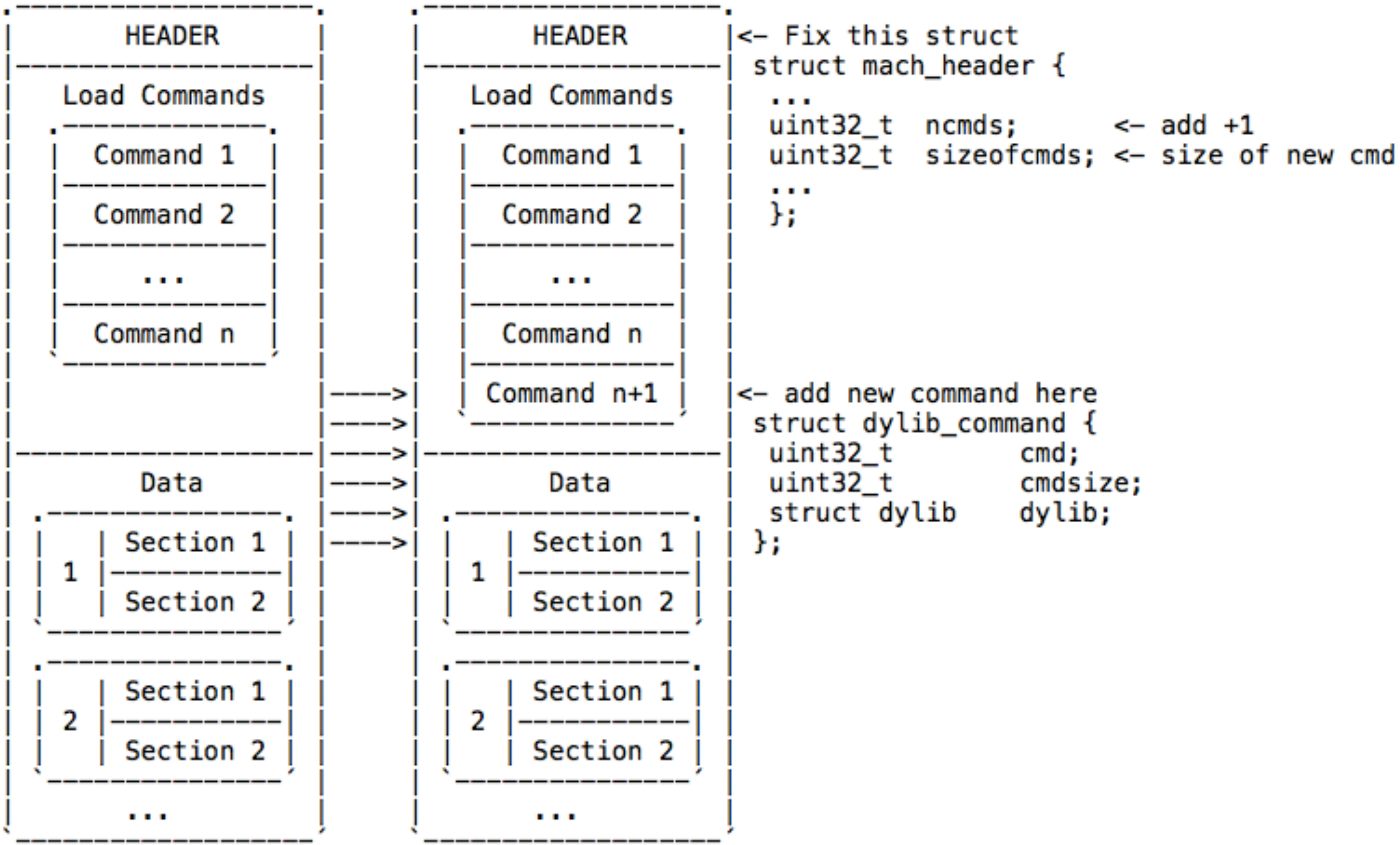
Version	Average Size	Min	Max
32bits	3013	28	49176
64bits	2601	32	36200

Minimum size required is 24bytes.
Check http://reverse.put.as/2012/01/31/anti-debug-trick-1-abusing-mach-o-to-crash-gdb/ for a complete description.

Code injection – How to do it

- Find the position of last segment command.
- Find the first data position, it’s either __text section or LC_ENCRYPTION_INFO (iOS).
- Calculate available space between the two.
- Add new command (if enough space available).
- Fix the header: size & nr of commands fields.
- Write/overwrite the new binary.

Code injection – How to do it



```
.-------------------.     .-------------------.
|       HEADER      |     |       HEADER      |<- Fix this struct
|-------------------|     |-------------------| struct mach_header {
|   Load Commands   |     |   Load Commands   |  ...
|  .-------------.  |     |  .-------------.  |  uint32_t  ncmds;      <- add +1
|  |  Command 1  |  |     |  |  Command 1  |  |  uint32_t  sizeofcmds; <- size of new cmd
|  |-------------|  |     |  |-------------|  |  ...
|  |  Command 2  |  |     |  |  Command 2  |  |  };
|  |-------------|  |     |  |-------------|  |
|  |     ...     |  |     |  |     ...     |  |
|  |-------------|  |     |  |-------------|  |
|  |  Command n  |  |     |  |  Command n  |  |
|  '-------------'  |     |  |-------------|  |
|                   |---->|  | Command n+1 |  |<- add new command here
|                   |---->|  '-------------'  | struct dylib_command {
|-------------------|---->|-------------------|  uint32_t        cmd;
|        Data       |---->|        Data       |  uint32_t        cmdsize;
| .---------------. |---->| .---------------. |  struct dylib    dylib;
| |   | Section 1 | |---->| |   | Section 1 | | };
| | 1 |-----------| |     | | 1 |-----------| |
| |   | Section 2 | |     | |   | Section 2 | |
| '---------------' |     | '---------------' |
| .---------------. |     | .---------------. |
| |   | Section 1 | |     | |   | Section 1 | |
| | 2 |-----------| |     | | 2 |-----------| |
| |   | Section 2 | |     | |   | Section 2 | |
| '---------------' |     | '---------------' |
|       ...         |     |       ...         |
'-------------------'     '-------------------'
```

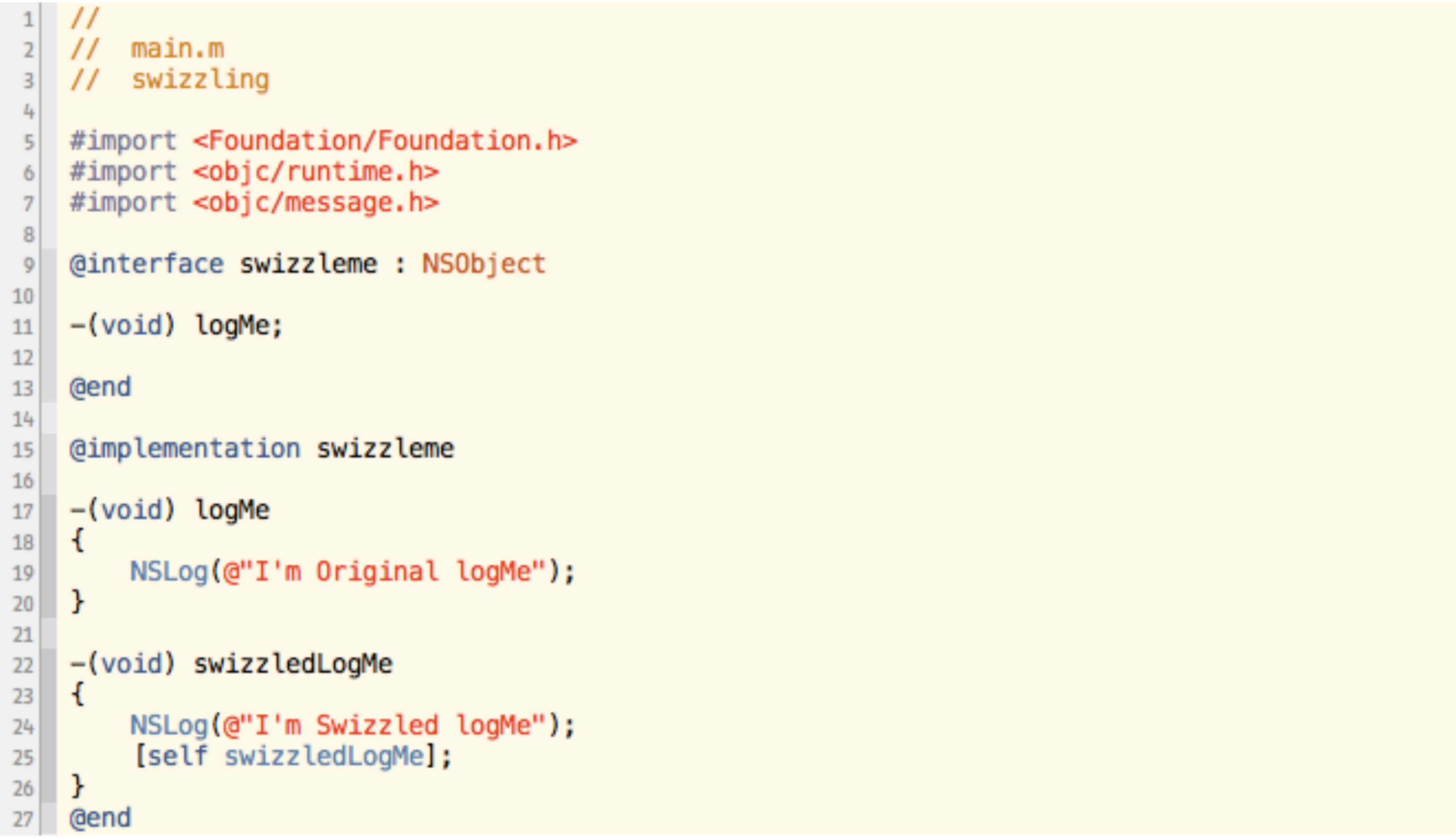
Code injection – How to do it

- Next step is to build a dynamic library.
- You can use the Xcode template.
- Add a constructor as the library entrypoint:
- extern void init(void) __attribute__ ((constructor));
- Do something.

Swizzling

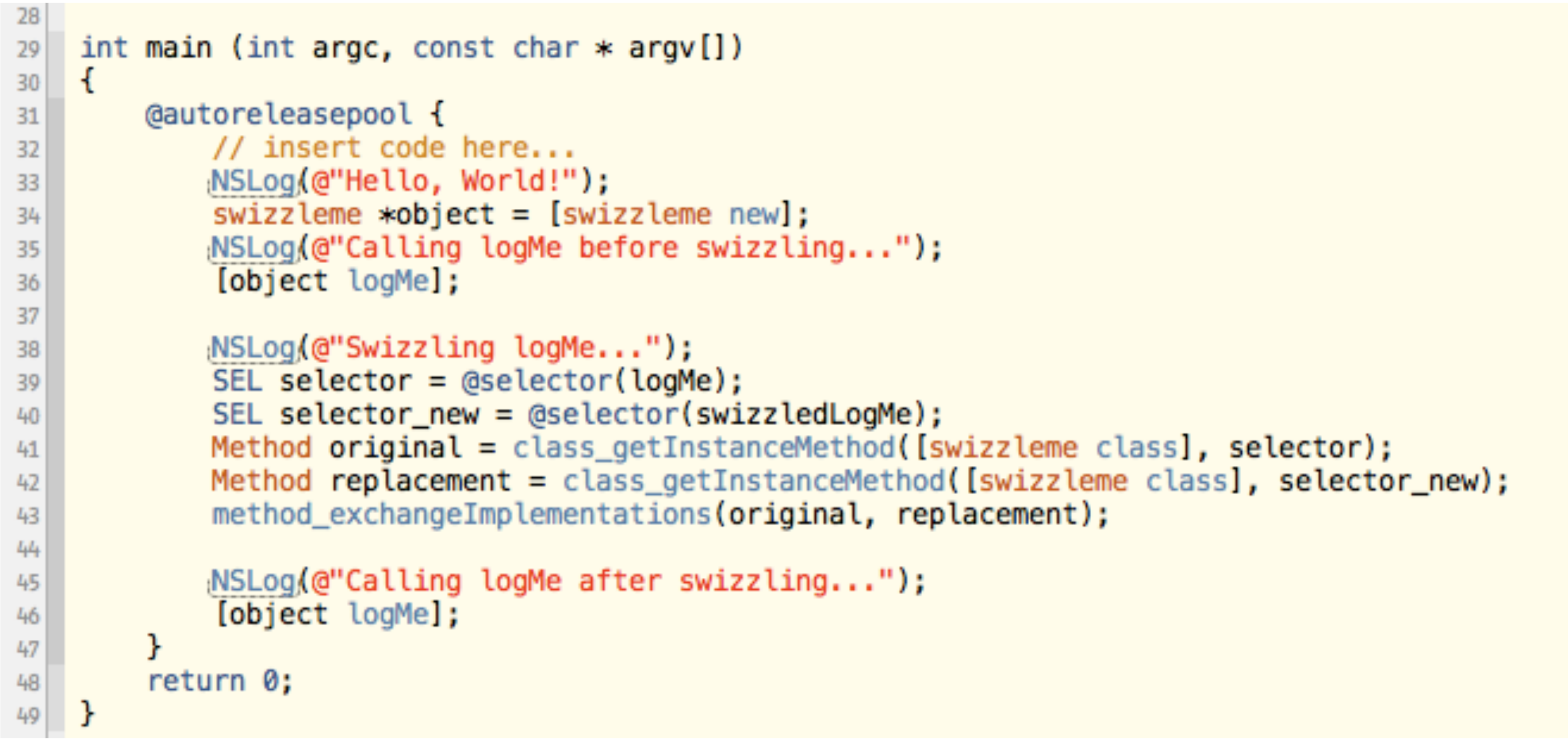
- Interesting Objective-C feature.
- Replace the method implementation with our own.
- We are still able to call the original selector.
- JRSwizzle makes this an easy process!
- Do whatever you want in your implementation:
- Dump credentials.
- Control access.
- Add features.
- Etc…

Swizzling – A basic example



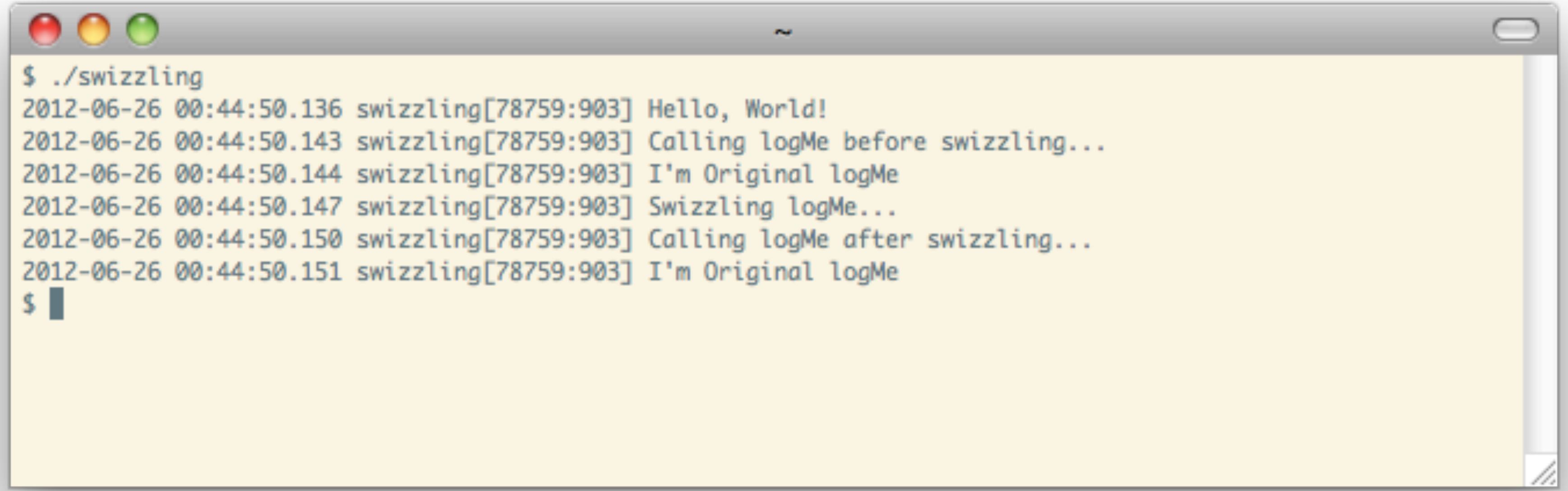
```
//
//  main.m
//  swizzling

#import <Foundation/Foundation.h>
#import <objc/runtime.h>
#import <objc/message.h>

@interface swizzleme : NSObject

-(void) logMe;

@end

@implementation swizzleme

-(void) logMe
{
    NSLog(@"I'm Original logMe");
}

-(void) swizzledLogMe
{
    NSLog(@"I'm Swizzled logMe");
    [self swizzledLogMe];
}
@end
```

Swizzling – A basic example



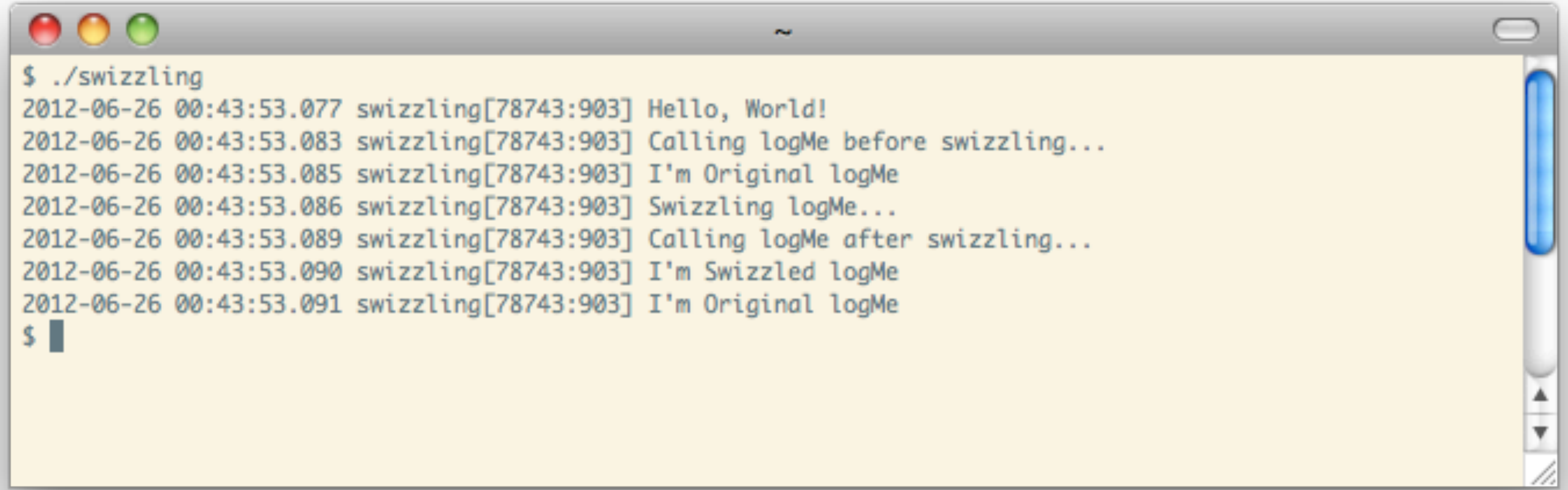
```
int main (int argc, const char * argv[])
{
    @autoreleasepool {
        // insert code here...
        NSLog(@"Hello, World!");
        swizzleme *object = [swizzleme new];
        NSLog(@"Calling logMe before swizzling...");
        [object logMe];

        NSLog(@"Swizzling logMe...");
        SEL selector = @selector(logMe);
        SEL selector_new = @selector(swizzledLogMe);
        Method original = class_getInstanceMethod([swizzleme class], selector);
        Method replacement = class_getInstanceMethod([swizzleme class], selector_new);
        method_exchangeImplementations(original, replacement);

        NSLog(@"Calling logMe after swizzling...");
        [object logMe];
    }
    return 0;
}
```

Swizzling – A basic example



```
$ ./swizzling
2012-06-26 00:44:50.136 swizzling[78759:903] Hello, World!
2012-06-26 00:44:50.143 swizzling[78759:903] Calling logMe before swizzling...
2012-06-26 00:44:50.144 swizzling[78759:903] I'm Original logMe
2012-06-26 00:44:50.147 swizzling[78759:903] Swizzling logMe...
2012-06-26 00:44:50.150 swizzling[78759:903] Calling logMe after swizzling...
2012-06-26 00:44:50.151 swizzling[78759:903] I'm Original logMe
$ 
```

Swizzling – A basic example



```
$ ./swizzling
2012-06-26 00:43:53.077 swizzling[78743:903] Hello, World!
2012-06-26 00:43:53.083 swizzling[78743:903] Calling logMe before swizzling...
2012-06-26 00:43:53.085 swizzling[78743:903] I'm Original logMe
2012-06-26 00:43:53.086 swizzling[78743:903] Swizzling logMe...
2012-06-26 00:43:53.089 swizzling[78743:903] Calling logMe after swizzling...
2012-06-26 00:43:53.090 swizzling[78743:903] I'm Swizzled logMe
2012-06-26 00:43:53.091 swizzling[78743:903] I'm Original logMe
$
```

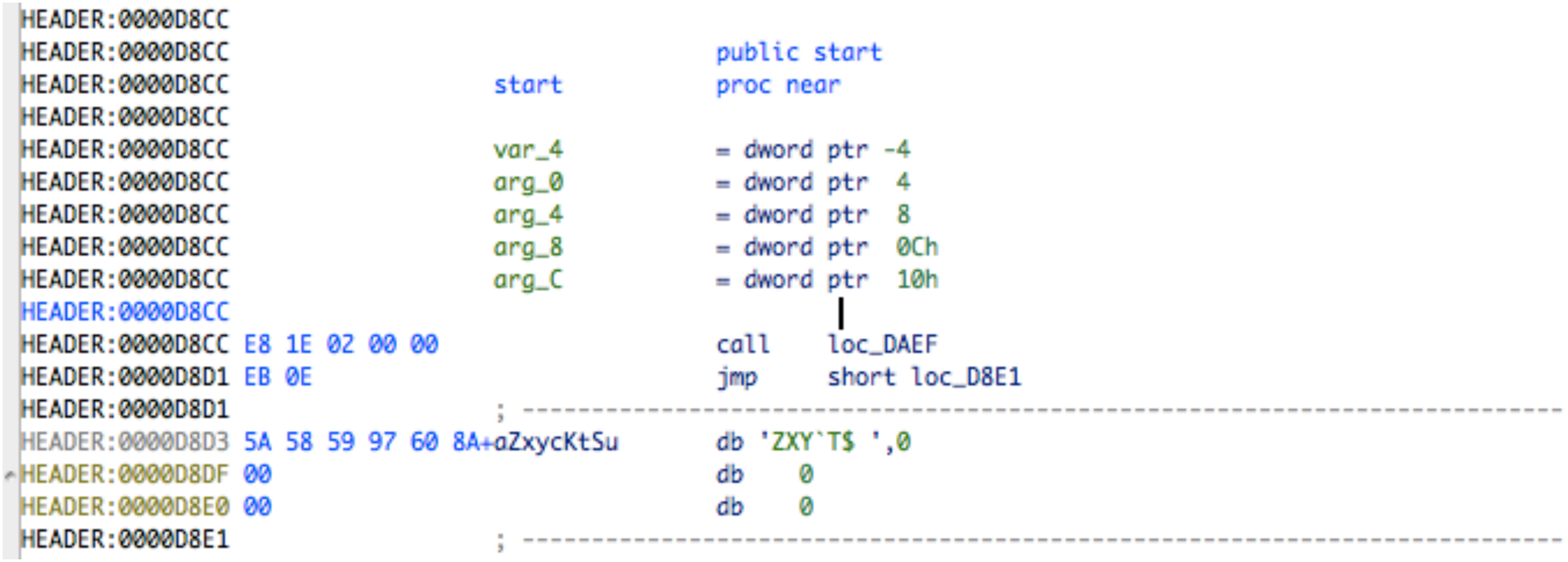
Tips & tricks – Packed binaries

- GDB doesn’t breakpoint entrypoint on packed binaries.
- My theory: this is due to abnormal Mach-O header.
- There’s only a __TEXT segment, without any sections.
- And a LC_UNIXTHREAD with the entrypoint.
- A workaround is to modify entrypoint and replace with INT3.
- And then manually fix things in GDB.
- Use my GDB patches to avoid a bug setting memory.

Tips & tricks – Packed binaries

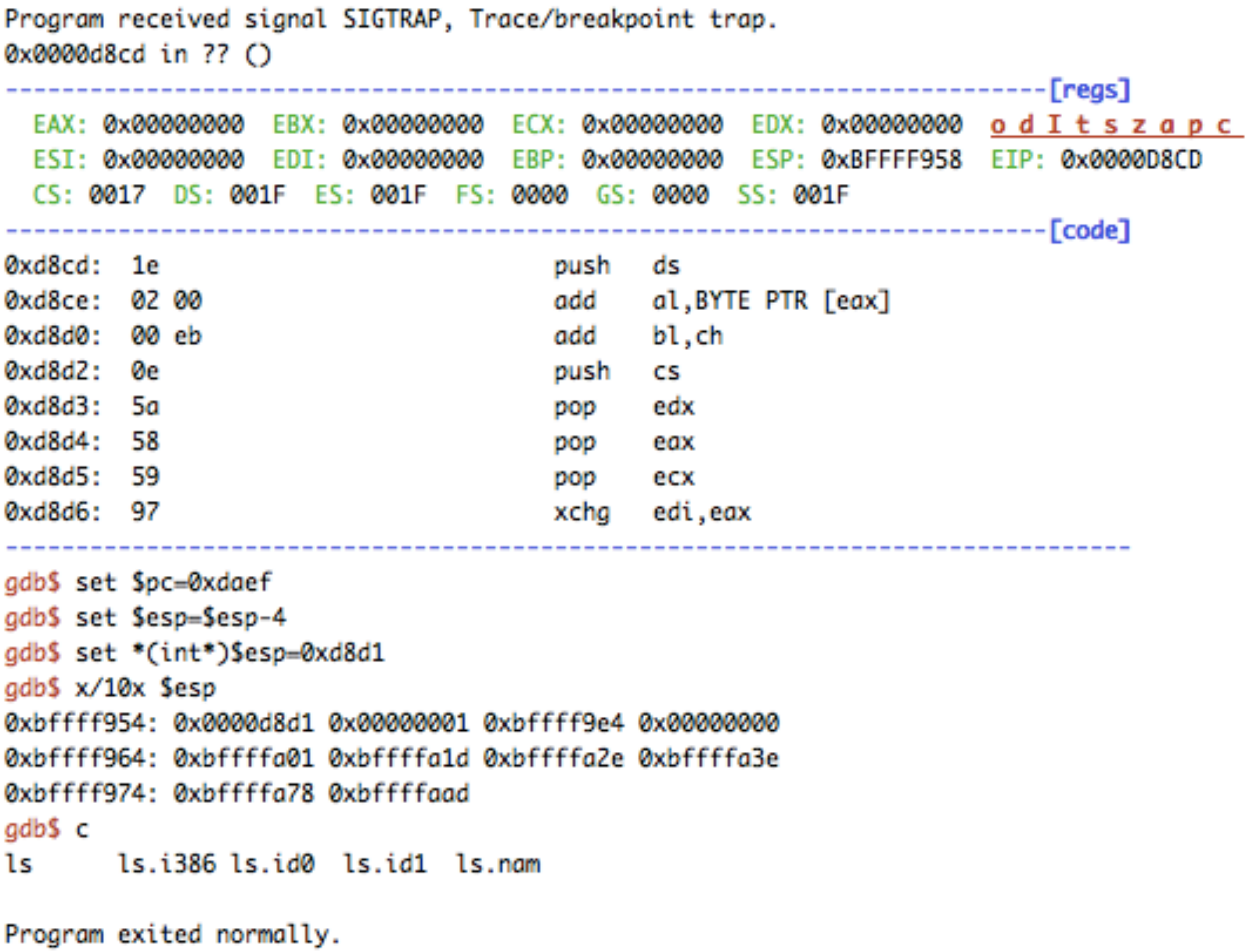
- In case of UPX, the entrypoint instruction is a call.
- So you will need to set the EIP to the correct address.
- Fix the stack pointer.
- And add the return address to the stack.
- Remove the INT3 and restore the original byte, to avoid checksum problems.
- Problems might occur if there's a secondary check between memory and disk image.

Tips & tricks – Packed binaries



```
HEADER:0000D8CC
HEADER:0000D8CC                                      public start
HEADER:0000D8CC                   start              proc near
HEADER:0000D8CC
HEADER:0000D8CC                   var_4              = dword ptr -4
HEADER:0000D8CC                   arg_0              = dword ptr  4
HEADER:0000D8CC                   arg_4              = dword ptr  8
HEADER:0000D8CC                   arg_8              = dword ptr  0Ch
HEADER:0000D8CC                   arg_C              = dword ptr  10h
HEADER:0000D8CC
HEADER:0000D8CC E8 1E 02 00 00                       call    loc_DAEF
HEADER:0000D8D1 EB 0E                                jmp     short loc_D8E1
HEADER:0000D8D1                   ; ---------------------------------------------------------------------------
HEADER:0000D8D3 5A 58 59 97 60 8A+aZxycKtSu          db 'ZXY`T$ ',0
HEADER:0000D8DF 00                                   db    0
HEADER:0000D8E0 00                                   db    0
HEADER:0000D8E1                   ; ---------------------------------------------------------------------------
```

Tips & tricks – Packed binaries

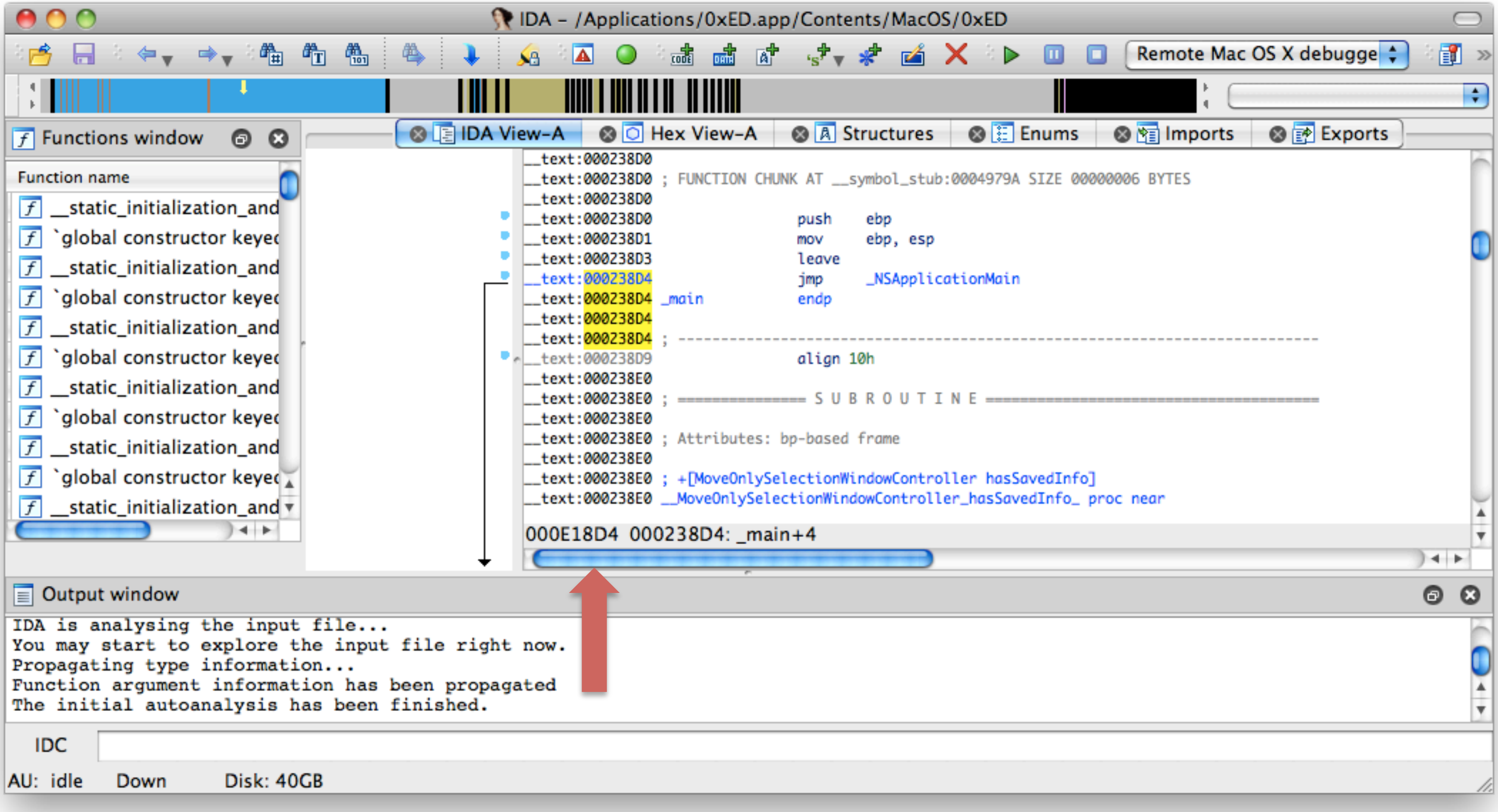


```
Program received signal SIGTRAP, Trace/breakpoint trap.
0x0000d8cd in ?? ()
-------------------------------------------------------------------------[regs]
  EAX: 0x00000000  EBX: 0x00000000  ECX: 0x00000000  EDX: 0x00000000  o d I t s z a p c
  ESI: 0x00000000  EDI: 0x00000000  EBP: 0x00000000  ESP: 0xBFFFF958  EIP: 0x0000D8CD
  CS: 0017  DS: 001F  ES: 001F  FS: 0000  GS: 0000  SS: 001F
-------------------------------------------------------------------------[code]
0xd8cd:  1e                           push   ds
0xd8ce:  02 00                        add    al,BYTE PTR [eax]
0xd8d0:  00 eb                        add    bl,ch
0xd8d2:  0e                           push   cs
0xd8d3:  5a                           pop    edx
0xd8d4:  58                           pop    eax
0xd8d5:  59                           pop    ecx
0xd8d6:  97                           xchg   edi,eax
-------------------------------------------------------------------------------
gdb$ set $pc=0xdaef
gdb$ set $esp=$esp-4
gdb$ set *(int*)$esp=0xd8d1
gdb$ x/10x $esp
0xbffff954: 0x0000d8d1 0x00000001 0xbffff9e4 0x00000000
0xbffff964: 0xbffffa01 0xbffffa1d 0xbffffa2e 0xbffffa3e
0xbffff974: 0xbffffa78 0xbffffaad
gdb$ c
ls      ls.i386 ls.id0  ls.id1  ls.nam

Program exited normally.
```

Tips & tricks – File offsets

- How to compute file offsets for patching:
- If you use IDA, the displayed offset is valid for fat and non-fat binaries.
- The vmaddr and fileoff fields on the next slides refer to the __TEXT segment.

Tips & tricks – File offsets



Function name

- __static_initialization_and
- `global constructor keyec
- __static_initialization_and
- `global constructor keyec
- __static_initialization_and
- `global constructor keyec
- __static_initialization_and
- `global constructor keyec
- __static_initialization_and
- `global constructor keyec
- __static_initialization_and

```
__text:000238D0
__text:000238D0 ; FUNCTION CHUNK AT __symbol_stub:0004979A SIZE 00000006 BYTES
__text:000238D0
__text:000238D0                 push    ebp
__text:000238D1                 mov     ebp, esp
__text:000238D3                 leave
__text:000238D4                 jmp     _NSApplicationMain
__text:000238D4 _main           endp
__text:000238D4
__text:000238D4 ; ---------------------------------------------------------------------------
__text:000238D9                 align 10h
__text:000238E0
__text:000238E0 ; =============== S U B R O U T I N E =======================================
__text:000238E0
__text:000238E0 ; Attributes: bp-based frame
__text:000238E0
__text:000238E0 ; +[MoveOnlySelectionWindowController hasSavedInfo]
__text:000238E0 __MoveOnlySelectionWindowController_hasSavedInfo_ proc near
```

000E18D4 000238D4: _main+4

```
IDA is analysing the input file...
You may start to explore the input file right now.
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.
```

Tips & tricks – File offsets

- Manually:
- 1) If binary is non-fat:

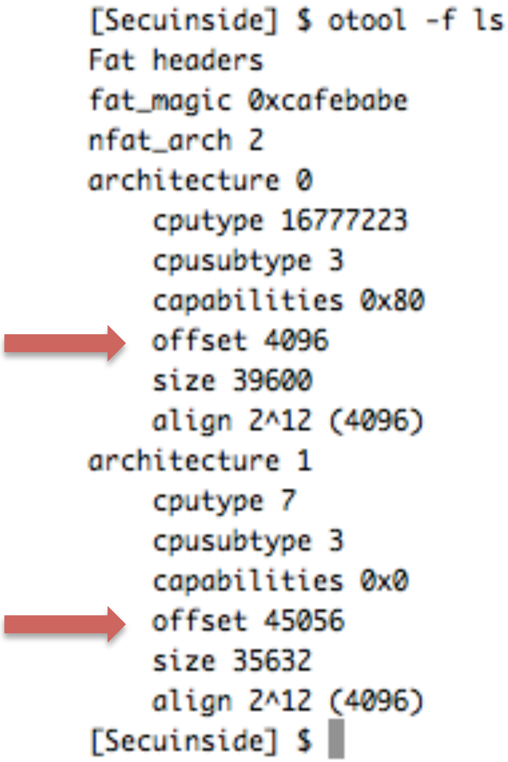
  File offset = Memory address - vmaddr + fileoff
- 2) If binary is fat:

  Retrieve offset of target arch from fat headers.

  File Offset = Target Arch Offset + Memory address - vmaddr + fileoff

Tips & tricks – File offsets

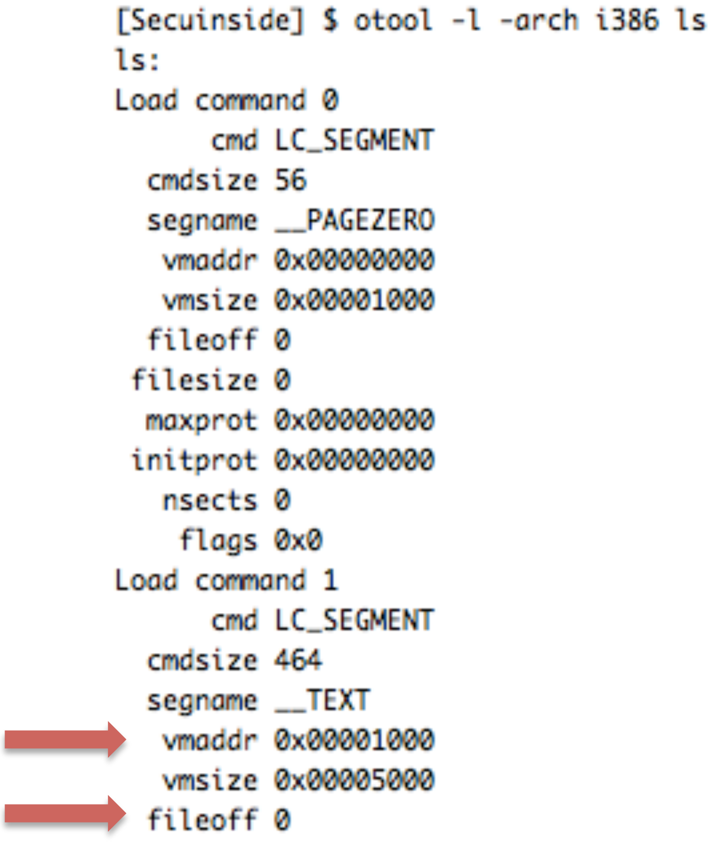
- Retrieve fat architecture file offset:



```
[Secuinside] $ otool -f ls
Fat headers
fat_magic 0xcafebabe
nfat_arch 2
architecture 0
    cputype 16777223
    cpusubtype 3
    capabilities 0x80
    offset 4096
    size 39600
    align 2^12 (4096)
architecture 1
    cputype 7
    cpusubtype 3
    capabilities 0x0
    offset 45056
    size 35632
    align 2^12 (4096)
[Secuinside] $
```

Tips & tricks – File offsets

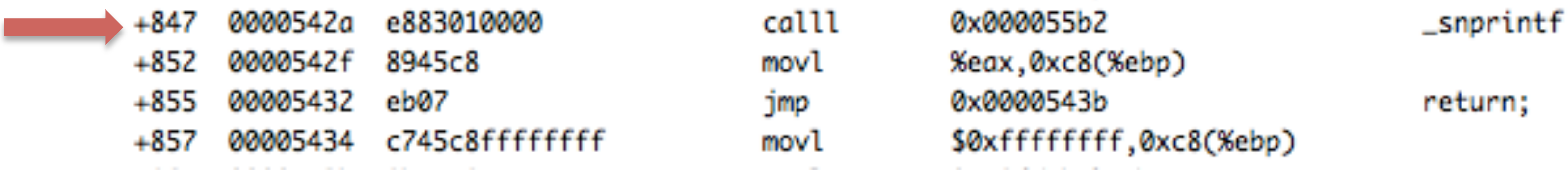
- Retrieve vmaddr and fileoff:



```
[Secuinside] $ otool -l -arch i386 ls
ls:
Load command 0
      cmd LC_SEGMENT
  cmdsize 56
  segname __PAGEZERO
   vmaddr 0x00000000
   vmsize 0x00001000
  fileoff 0
 filesize 0
  maxprot 0x00000000
 initprot 0x00000000
   nsects 0
    flags 0x0
Load command 1
      cmd LC_SEGMENT
  cmdsize 464
  segname __TEXT
   vmaddr 0x00001000
   vmsize 0x00005000
  fileoff 0
```

Tips & tricks – File offsets

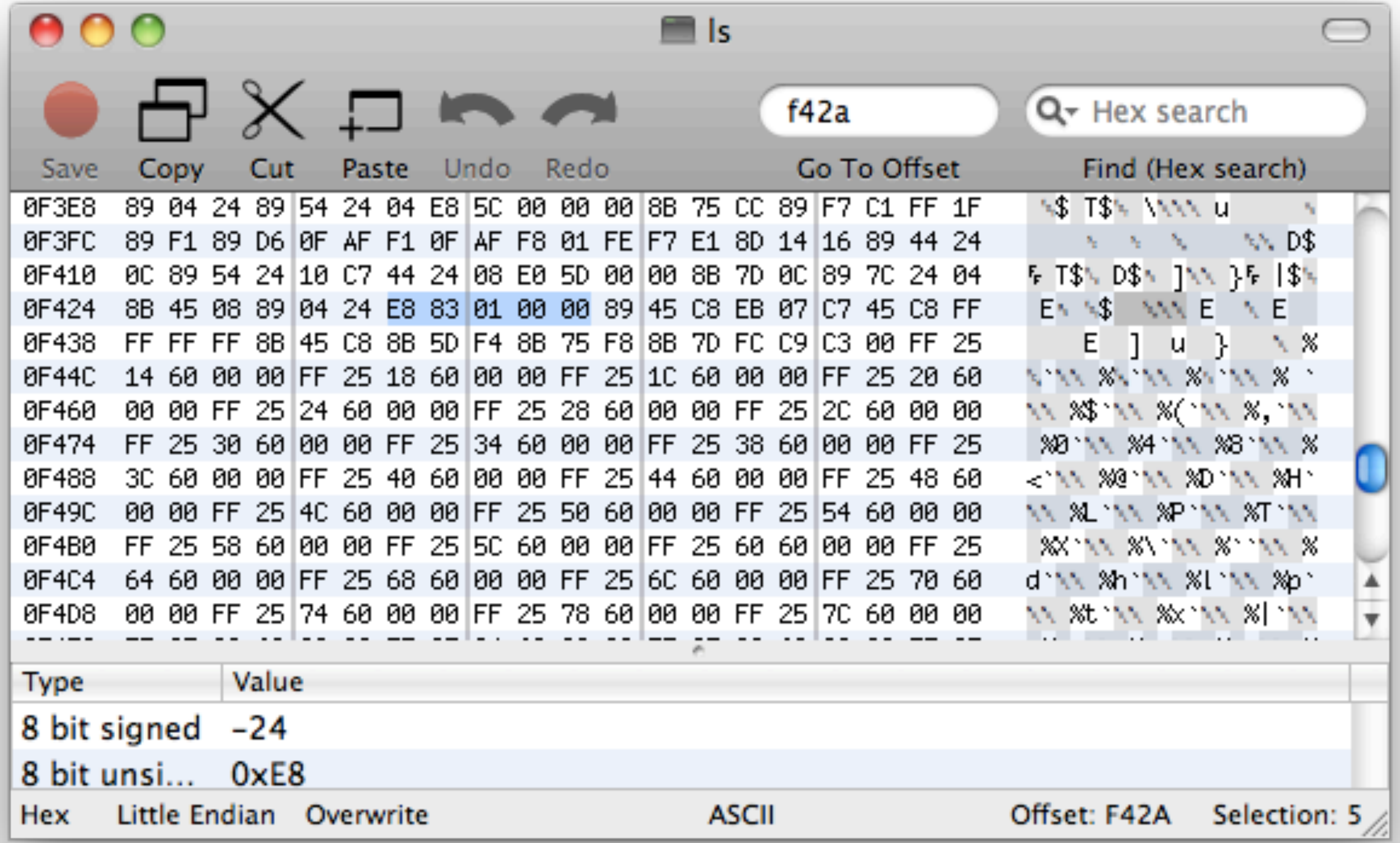
- Calculate the file offset for a given address:



```
+847  0000542a  e883010000         calll   0x000055b2              _snprintf
+852  0000542f  8945c8             movl    %eax,0xc8(%ebp)
+855  00005432  eb07               jmp     0x0000543b              return;
+857  00005434  c745c8ffffffff     movl    $0xffffffff,0xc8(%ebp)
```

File offset = 45056 + 0x542a – 0x1000 + 0 = 0xF42A

Tips & tricks – File offsets



f42a

```
0F3E8   89 04 24 89 54 24 04 E8 5C 00 00 00 8B 75 CC 89 F7 C1 FF 1F
0F3FC   89 F1 89 D6 0F AF F1 0F AF F8 01 FE F7 E1 8D 14 16 89 44 24
0F410   0C 89 54 24 10 C7 44 24 08 E0 5D 00 00 8B 7D 0C 89 7C 24 04
0F424   8B 45 08 89 04 24 E8 83 01 00 00 89 45 C8 EB 07 C7 45 C8 FF
0F438   FF FF FF 8B 45 C8 8B 5D F4 8B 75 F8 8B 7D FC C9 C3 00 FF 25
0F44C   14 60 00 00 FF 25 18 60 00 00 FF 25 1C 60 00 00 FF 25 20 60
0F460   00 00 FF 25 24 60 00 00 FF 25 28 60 00 00 FF 25 2C 60 00 00
0F474   FF 25 30 60 00 00 FF 25 34 60 00 00 FF 25 38 60 00 00 FF 25
0F488   3C 60 00 00 FF 25 40 60 00 00 FF 25 44 60 00 00 FF 25 48 60
0F49C   00 00 FF 25 4C 60 00 00 FF 25 50 60 00 00 FF 25 54 60 00 00
0F4B0   FF 25 58 60 00 00 FF 25 5C 60 00 00 FF 25 60 60 00 00 FF 25
0F4C4   64 60 00 00 FF 25 68 60 00 00 FF 25 6C 60 00 00 FF 25 70 60
0F4D8   00 00 FF 25 74 60 00 00 FF 25 78 60 00 00 FF 25 7C 60 00 00
```

Type	Value
8 bit signed	-24
8 bit unsi...	0xE8

Hex   Little Endian   Overwrite   ASCII   Offset: F42A   Selection: 5

Tips & tricks – Resigning binaries

- Code signing introduced in Leopard.
- In practice it’s useless. Barely any app uses it in a proper way.
- We can patch the app and resign it with our own certificate.
- Of course, assuming no certificate validation (never saw an app that does it!).

Tips & tricks – Resigning binaries

- Generate your self-signed code signing certificate.
- Using Certificate Assistant of Keychain app.
- Or by hand with OpenSSL [22].
- Resign the modified application:
- codesign –s “cert_name” –vvvv –f target_binary
- Or just remove LC_CODE_SIGNATURE from Mach-O header.

iOS Reversing

- Almost all of the previous slides apply.
- If your target is armv7, you will have some problems.
- GDB is unable to correctly disassemble some instructions, so output is all messed up.
- My method is to follow code in IDA, while stepping in GDB (yes, it sucks!).
- Hopper author is working on ARM support and will implement a debug server for iOS.

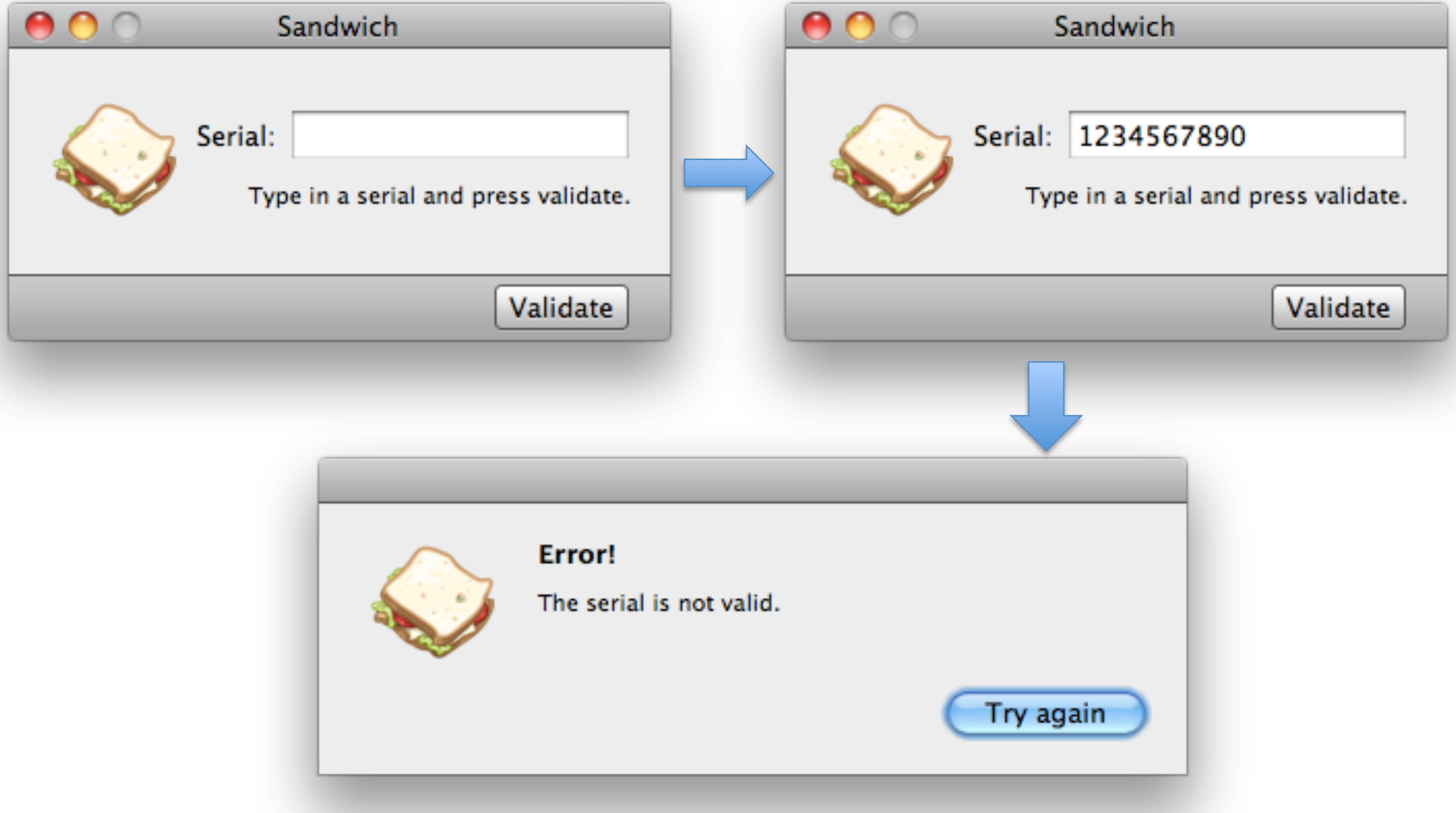
iOS Reversing

- If you want to overwrite iOS binaries, don’t forget that inodes must change.
- Just mv or rm the original file and copy the new/patched file.
- ldone from hackulo.us repo works great for fake code signing.
- Cydia.radare.org repo has an updated GDB version with my patches.

Reversing a crackme

- Target is Sandwich.
- A very simple and rather old Cocoa crackme.
- Available at http://reverse.put.as/wp-content/uploads/2010/05/1-Sandwich.zip.
- A couple more crackmes available at http://reverse.put.as/crackmes/.
- Try to reverse my crackme, it uses some interesting tricks ☺.

Reversing a crackme



Serial:

Type in a serial and press validate.

Serial: 1234567890

Type in a serial and press validate.

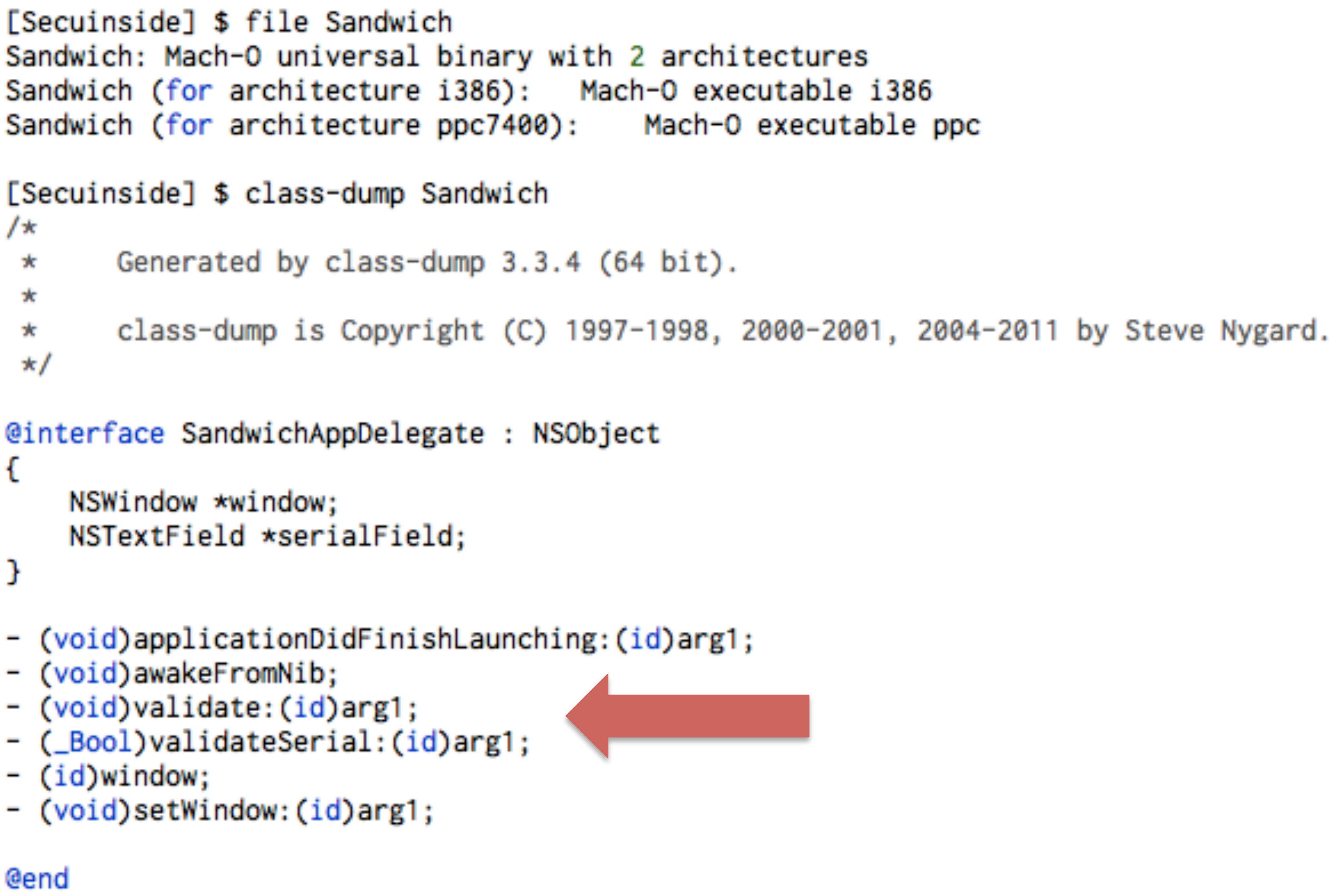
Error!

The serial is not valid.

Reversing a crackme

- What is inside?
- We can start by using the file command to verify the available architectures.
- And then use class-dump to dump methods.
- Or use nm to display the symbols.
- I also like to use otool –l (or MachOView) to have a look at the Mach-O load commands.
- It allows you to spot unusual stuff.

Reversing a crackme

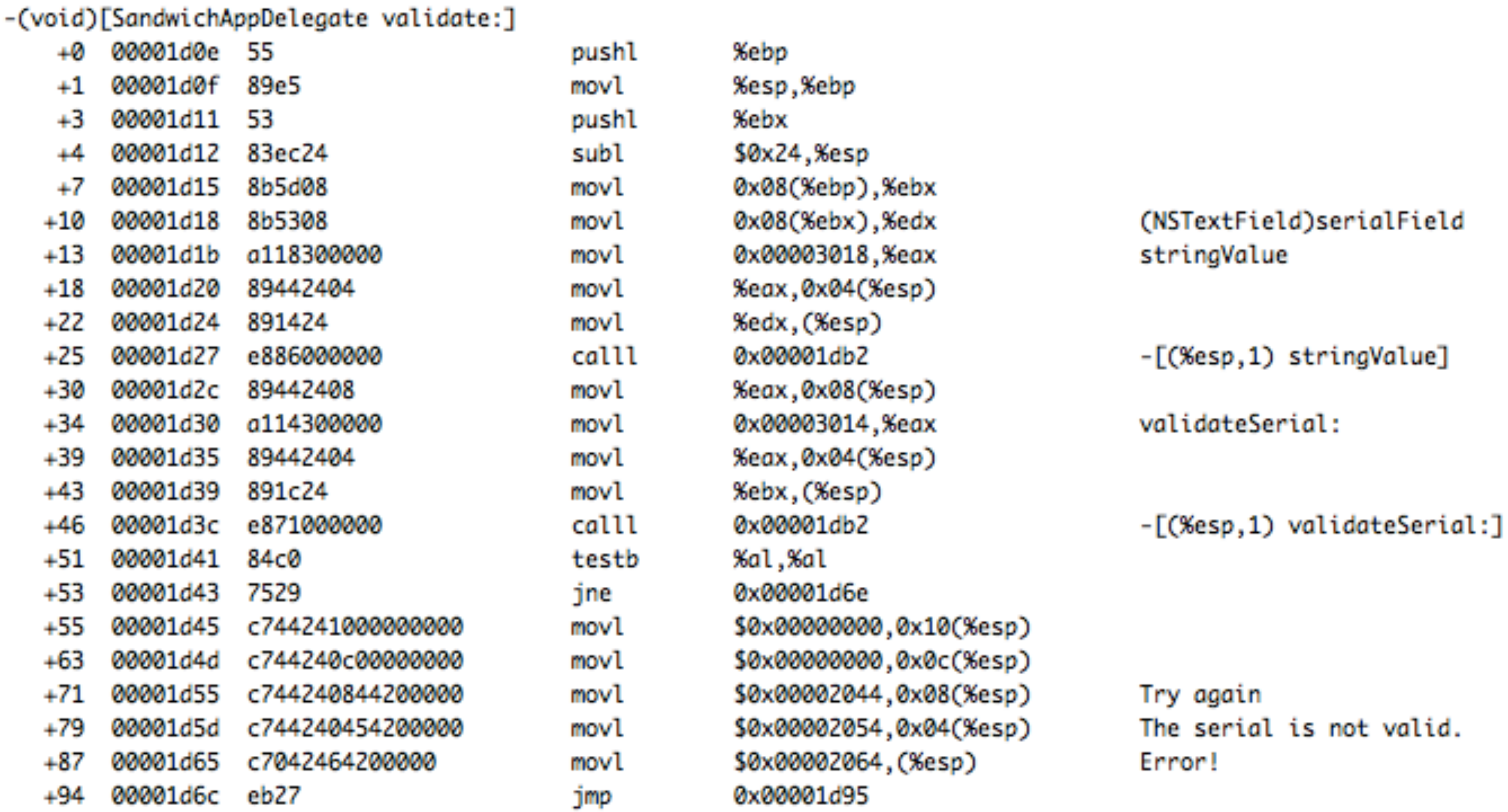


```
[Secuinside] $ file Sandwich
Sandwich: Mach-O universal binary with 2 architectures
Sandwich (for architecture i386):   Mach-O executable i386
Sandwich (for architecture ppc7400):    Mach-O executable ppc

[Secuinside] $ class-dump Sandwich
/*
 *     Generated by class-dump 3.3.4 (64 bit).
 *
 *     class-dump is Copyright (C) 1997-1998, 2000-2001, 2004-2011 by Steve Nygard.
 */

@interface SandwichAppDelegate : NSObject
{
    NSWindow *window;
    NSTextField *serialField;
}

- (void)applicationDidFinishLaunching:(id)arg1;
- (void)awakeFromNib;
- (void)validate:(id)arg1;
- (_Bool)validateSerial:(id)arg1;
- (id)window;
- (void)setWindow:(id)arg1;

@end
```

Reversing a crackme

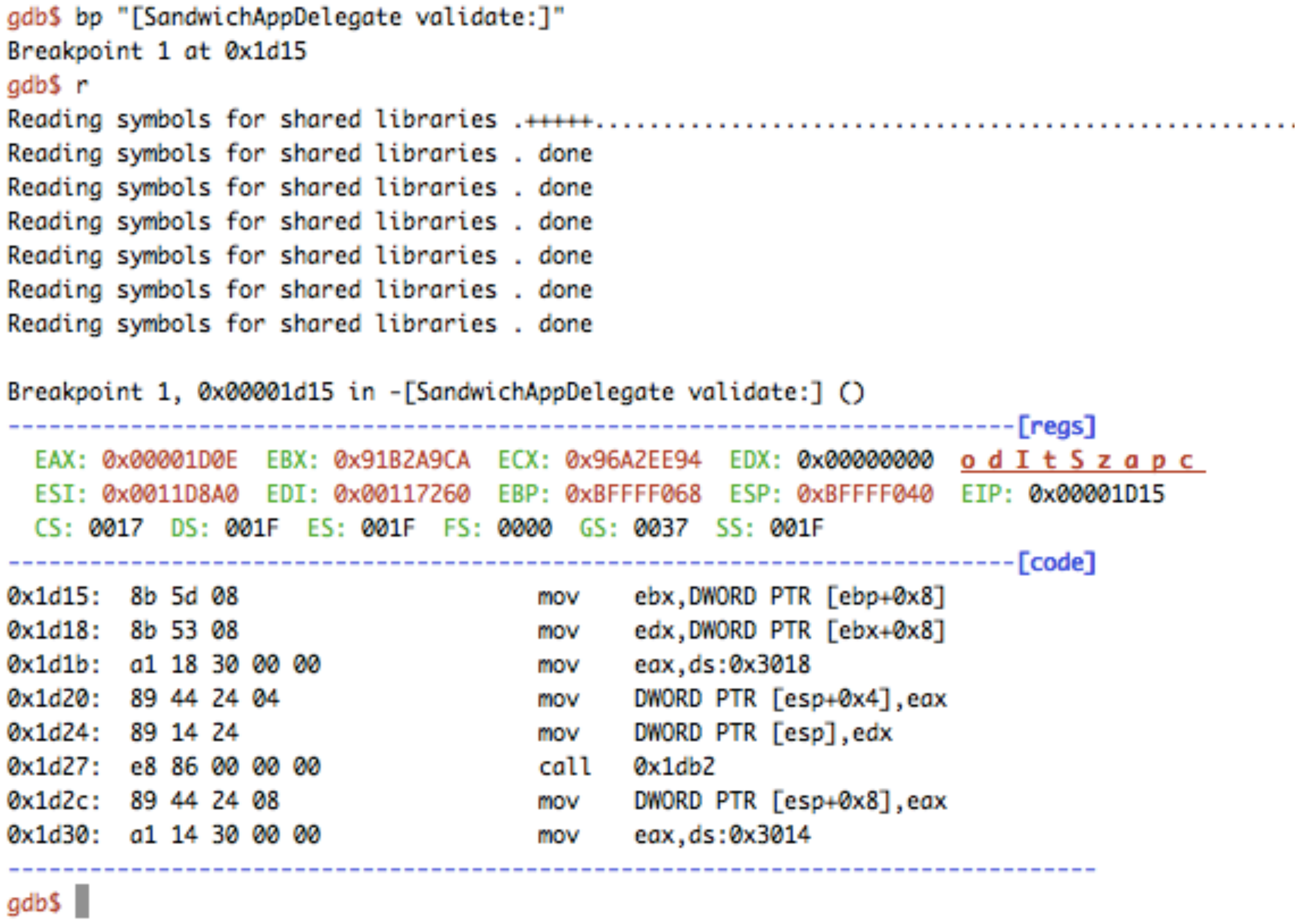
- The methods validate: and validateSerial: have appealing names.
- We can disassemble the binary and give a look at those methods.
- In this example I used OTX command line version.
- And we can also use GDB to verify if those methods are used or not.

Reversing a crackme



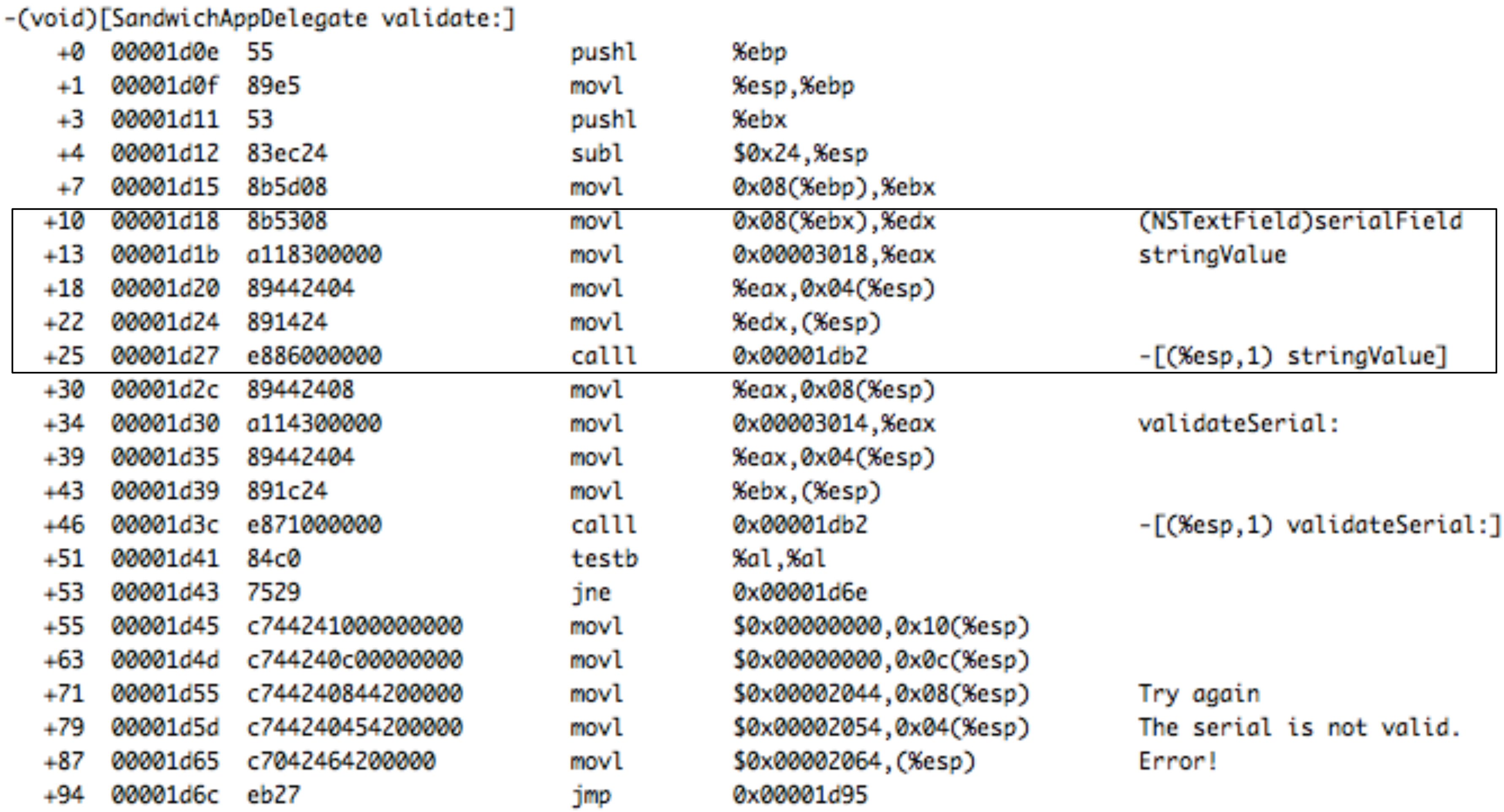
```
-(void)[SandwichAppDelegate validate:]
    +0  00001d0e  55                     pushl     %ebp
    +1  00001d0f  89e5                   movl      %esp,%ebp
    +3  00001d11  53                     pushl     %ebx
    +4  00001d12  83ec24                 subl      $0x24,%esp
    +7  00001d15  8b5d08                 movl      0x08(%ebp),%ebx
   +10  00001d18  8b5308                 movl      0x08(%ebx),%edx              (NSTextField)serialField
   +13  00001d1b  a118300000             movl      0x00003018,%eax              stringValue
   +18  00001d20  89442404               movl      %eax,0x04(%esp)
   +22  00001d24  891424                 movl      %edx,(%esp)
   +25  00001d27  e886000000             calll     0x00001db2                   -[(%esp,1) stringValue]
   +30  00001d2c  89442408               movl      %eax,0x08(%esp)
   +34  00001d30  a114300000             movl      0x00003014,%eax              validateSerial:
   +39  00001d35  89442404               movl      %eax,0x04(%esp)
   +43  00001d39  891c24                 movl      %ebx,(%esp)
   +46  00001d3c  e871000000             calll     0x00001db2                   -[(%esp,1) validateSerial:]
   +51  00001d41  84c0                   testb     %al,%al
   +53  00001d43  7529                   jne       0x00001d6e
   +55  00001d45  c744241000000000       movl      $0x00000000,0x10(%esp)
   +63  00001d4d  c744240c00000000       movl      $0x00000000,0x0c(%esp)
   +71  00001d55  c744240844200000       movl      $0x00002044,0x08(%esp)       Try again
   +79  00001d5d  c744240454200000       movl      $0x00002054,0x04(%esp)       The serial is not valid.
   +87  00001d65  c7042464200000         movl      $0x00002064,(%esp)           Error!
   +94  00001d6c  eb27                   jmp       0x00001d95
```

Reversing a crackme



```
gdb$ bp "[SandwichAppDelegate validate:]"
Breakpoint 1 at 0x1d15
gdb$ r
Reading symbols for shared libraries .+++++..........................................
Reading symbols for shared libraries . done
Reading symbols for shared libraries . done
Reading symbols for shared libraries . done
Reading symbols for shared libraries . done
Reading symbols for shared libraries . done
Reading symbols for shared libraries . done

Breakpoint 1, 0x00001d15 in -[SandwichAppDelegate validate:] ()
-----------------------------------------------------------------------[regs]
  EAX: 0x00001D0E  EBX: 0x91B2A9CA  ECX: 0x96A2EE94  EDX: 0x00000000  o d I t S z a p c
  ESI: 0x0011D8A0  EDI: 0x00117260  EBP: 0xBFFFF068  ESP: 0xBFFFF040  EIP: 0x00001D15
  CS: 0017  DS: 001F  ES: 001F  FS: 0000  GS: 0037  SS: 001F
-----------------------------------------------------------------------[code]
0x1d15:  8b 5d 08                     mov    ebx,DWORD PTR [ebp+0x8]
0x1d18:  8b 53 08                     mov    edx,DWORD PTR [ebx+0x8]
0x1d1b:  a1 18 30 00 00               mov    eax,ds:0x3018
0x1d20:  89 44 24 04                  mov    DWORD PTR [esp+0x4],eax
0x1d24:  89 14 24                     mov    DWORD PTR [esp],edx
0x1d27:  e8 86 00 00 00               call   0x1db2
0x1d2c:  89 44 24 08                  mov    DWORD PTR [esp+0x8],eax
0x1d30:  a1 14 30 00 00               mov    eax,ds:0x3014
-----------------------------------------------------------------------------
gdb$ 
```

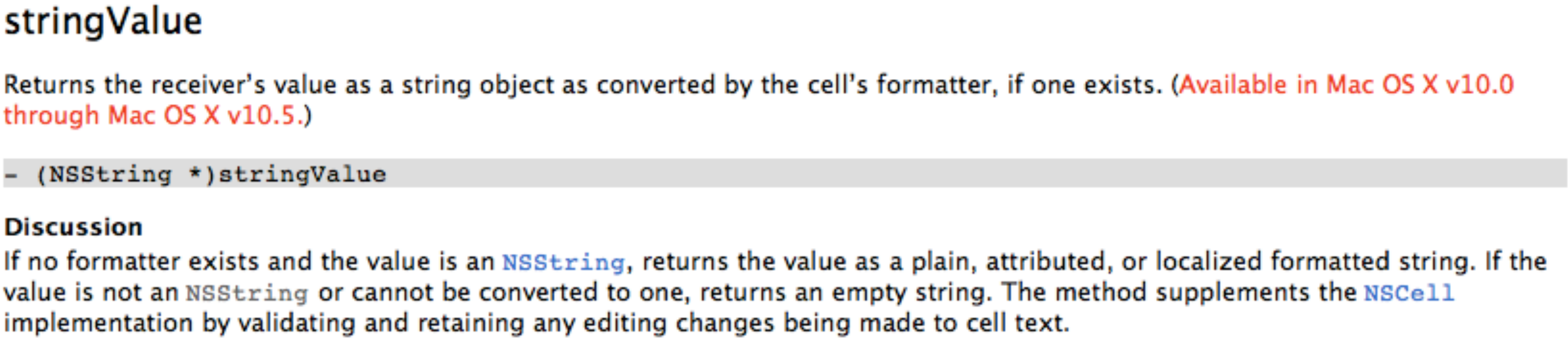
Reversing a crackme



```
-(void)[SandwichAppDelegate validate:]
   +0  00001d0e  55                pushl   %ebp
   +1  00001d0f  89e5              movl    %esp,%ebp
   +3  00001d11  53                pushl   %ebx
   +4  00001d12  83ec24            subl    $0x24,%esp
   +7  00001d15  8b5d08            movl    0x08(%ebp),%ebx
  +10  00001d18  8b5308            movl    0x08(%ebx),%edx             (NSTextField)serialField
  +13  00001d1b  a118300000        movl    0x00003018,%eax             stringValue
  +18  00001d20  89442404          movl    %eax,0x04(%esp)
  +22  00001d24  891424            movl    %edx,(%esp)
  +25  00001d27  e886000000        calll   0x00001db2                  -[(%esp,1) stringValue]
  +30  00001d2c  89442408          movl    %eax,0x08(%esp)
  +34  00001d30  a114300000        movl    0x00003014,%eax             validateSerial:
  +39  00001d35  89442404          movl    %eax,0x04(%esp)
  +43  00001d39  891c24            movl    %ebx,(%esp)
  +46  00001d3c  e871000000        calll   0x00001db2                  -[(%esp,1) validateSerial:]
  +51  00001d41  84c0              testb   %al,%al
  +53  00001d43  7529              jne     0x00001d6e
  +55  00001d45  c744241000000000  movl    $0x00000000,0x10(%esp)
  +63  00001d4d  c744240c00000000  movl    $0x00000000,0x0c(%esp)
  +71  00001d55  c744240844200000  movl    $0x00002044,0x08(%esp)      Try again
  +79  00001d5d  c744240454200000  movl    $0x00002054,0x04(%esp)      The serial is not valid.
  +87  00001d65  c7042464200000    movl    $0x00002064,(%esp)          Error!
  +94  00001d6c  eb27              jmp     0x00001d95
```

Reversing a crackme

- The stringValue method is retrieving the serial number we input into the box.
- Browsing documentation in Xcode or Dash we have:



stringValue

Returns the receiver's value as a string object as converted by the cell's formatter, if one exists. (Available in Mac OS X v10.0 through Mac OS X v10.5.)

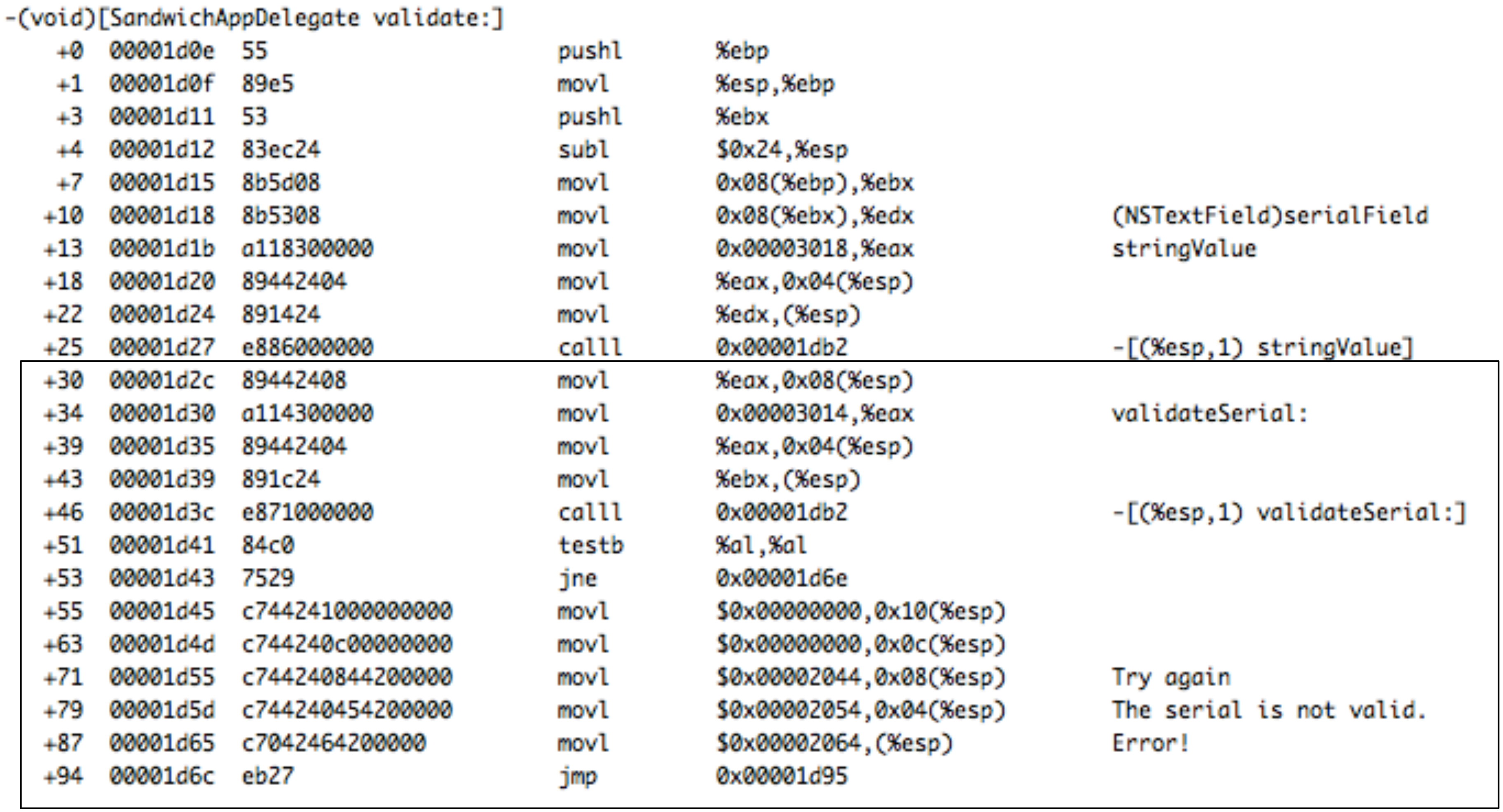
```
- (NSString *)stringValue
```

Discussion

If no formatter exists and the value is an NSString, returns the value as a plain, attributed, or localized formatted string. If the value is not an NSString or cannot be converted to one, returns an empty string. The method supplements the NSCell implementation by validating and retaining any editing changes being made to cell text.

- Returns a string object with that NSTextField contents.

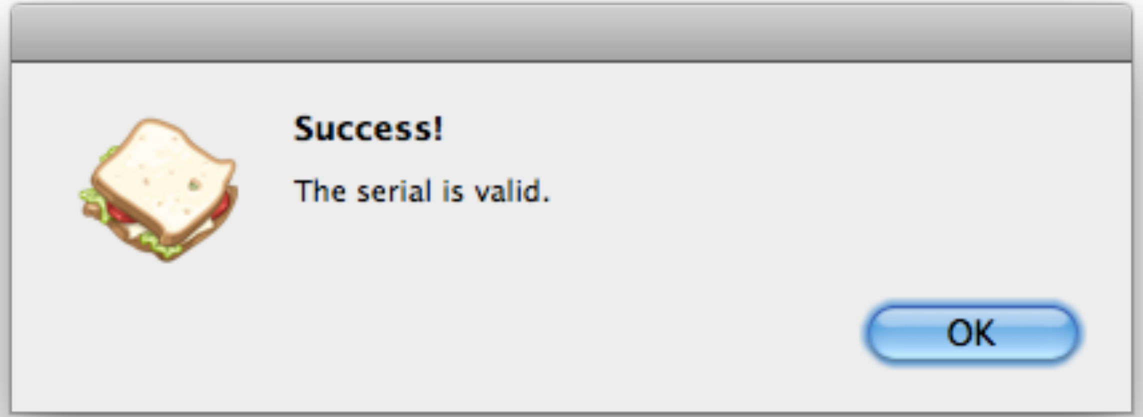
Reversing a crackme



```
-(void)[SandwichAppDelegate validate:]
    +0  00001d0e  55                 pushl   %ebp
    +1  00001d0f  89e5               movl    %esp,%ebp
    +3  00001d11  53                 pushl   %ebx
    +4  00001d12  83ec24             subl    $0x24,%esp
    +7  00001d15  8b5d08             movl    0x08(%ebp),%ebx
   +10  00001d18  8b5308             movl    0x08(%ebx),%edx                (NSTextField)serialField
   +13  00001d1b  a118300000         movl    0x00003018,%eax                stringValue
   +18  00001d20  89442404           movl    %eax,0x04(%esp)
   +22  00001d24  891424             movl    %edx,(%esp)
   +25  00001d27  e886000000         calll   0x00001db2                     -[(%esp,1) stringValue]
   +30  00001d2c  89442408           movl    %eax,0x08(%esp)
   +34  00001d30  a114300000         movl    0x00003014,%eax                validateSerial:
   +39  00001d35  89442404           movl    %eax,0x04(%esp)
   +43  00001d39  891c24             movl    %ebx,(%esp)
   +46  00001d3c  e871000000         calll   0x00001db2                     -[(%esp,1) validateSerial:]
   +51  00001d41  84c0               testb   %al,%al
   +53  00001d43  7529               jne     0x00001d6e
   +55  00001d45  c744241000000000   movl    $0x00000000,0x10(%esp)
   +63  00001d4d  c744240c00000000   movl    $0x00000000,0x0c(%esp)
   +71  00001d55  c744240844200000   movl    $0x00002044,0x08(%esp)         Try again
   +79  00001d5d  c744240454200000   movl    $0x00002054,0x04(%esp)         The serial is not valid.
   +87  00001d65  c7042464200000     movl    $0x00002064,(%esp)             Error!
   +94  00001d6c  eb27               jmp     0x00001d95
```

Reversing a crackme

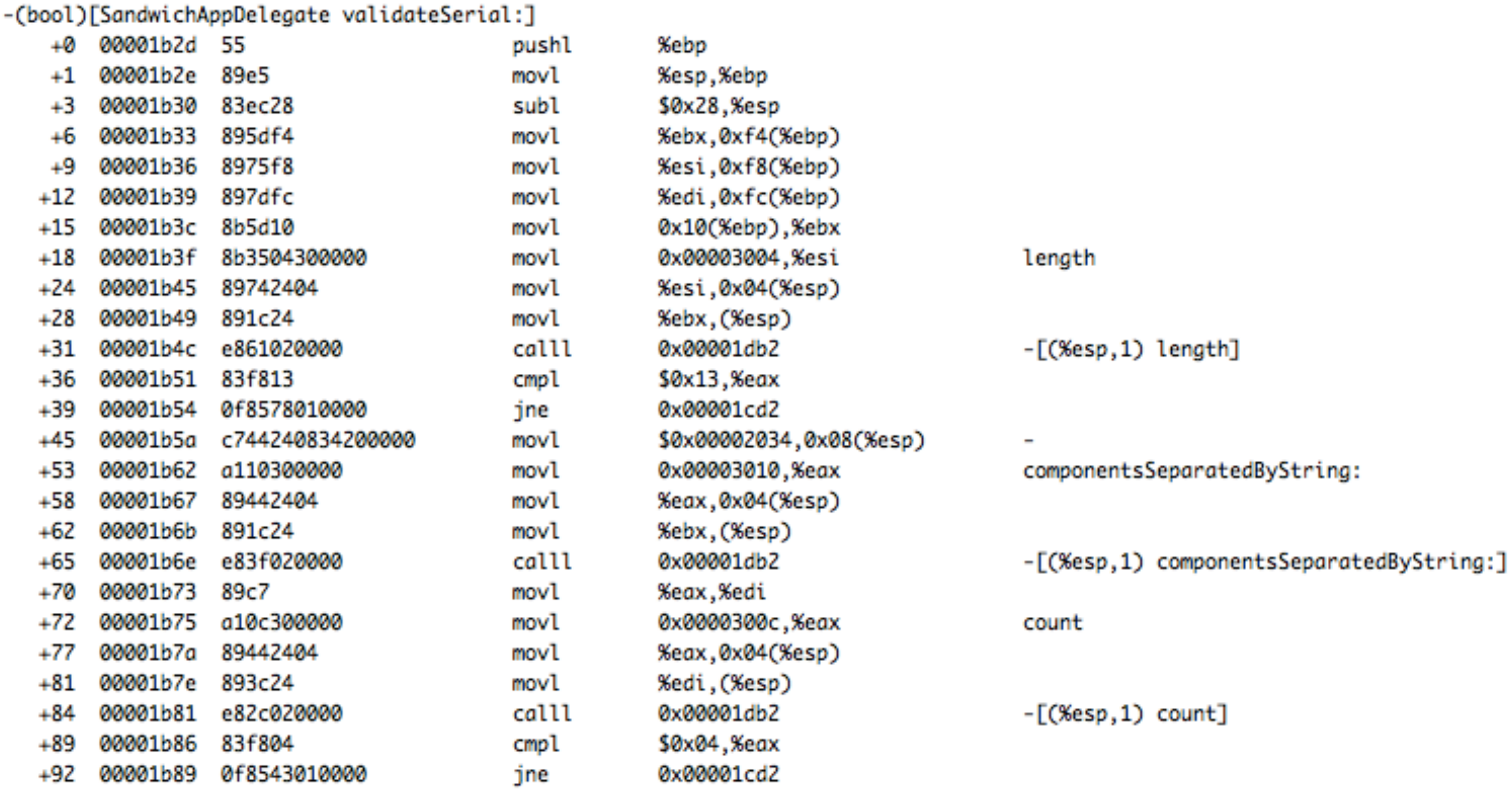
- The method validateSerial: is called with the serial number as the only argument.
- Returns a bool with success or failure.
- If we modify the JNE at 0x1d43:



Success!

The serial is valid.

Reversing a crackme



```
-(bool)[SandwichAppDelegate validateSerial:]
    +0  00001b2d  55                  pushl   %ebp
    +1  00001b2e  89e5                movl    %esp,%ebp
    +3  00001b30  83ec28              subl    $0x28,%esp
    +6  00001b33  895df4              movl    %ebx,0xf4(%ebp)
    +9  00001b36  8975f8              movl    %esi,0xf8(%ebp)
   +12  00001b39  897dfc              movl    %edi,0xfc(%ebp)
   +15  00001b3c  8b5d10              movl    0x10(%ebp),%ebx
   +18  00001b3f  8b3504300000        movl    0x00003004,%esi             length
   +24  00001b45  89742404            movl    %esi,0x04(%esp)
   +28  00001b49  891c24              movl    %ebx,(%esp)
   +31  00001b4c  e861020000          calll   0x00001db2                  -[(%esp,1) length]
   +36  00001b51  83f813              cmpl    $0x13,%eax
   +39  00001b54  0f8578010000        jne     0x00001cd2
   +45  00001b5a  c744240834200000    movl    $0x00002034,0x08(%esp)      -
   +53  00001b62  a110300000          movl    0x00003010,%eax             componentsSeparatedByString:
   +58  00001b67  89442404            movl    %eax,0x04(%esp)
   +62  00001b6b  891c24              movl    %ebx,(%esp)
   +65  00001b6e  e83f020000          calll   0x00001db2                  -[(%esp,1) componentsSeparatedByString:]
   +70  00001b73  89c7                movl    %eax,%edi
   +72  00001b75  a10c300000          movl    0x0000300c,%eax             count
   +77  00001b7a  89442404            movl    %eax,0x04(%esp)
   +81  00001b7e  893c24              movl    %edi,(%esp)
   +84  00001b81  e82c020000          calll   0x00001db2                  -[(%esp,1) count]
   +89  00001b86  83f804              cmpl    $0x04,%eax
   +92  00001b89  0f8543010000        jne     0x00001cd2
```

Reversing a crackme

- Now it's a matter of following the code and reversing the serial algorithm.
- Length should be 19 chars.
- It should contain 4 groups of characters separated by a dash (-).
- And so on…
- You should be able to follow what is happening by checking methods documentation.

Final remarks

- OS X is an interesting platform.
- Lags in both offensive and defensive reversing tools & tricks, especially if compared with Windows.
- This is great for all of you that like to do research!
- Not so crowded space as Windows and Linux.
- Lots of opportunities to create new things.
- And hopefully to do interesting presentations ;-).

References

1. http://reverse.put.as
2. http://ho.ax
3. http://www.phrack.org/issues.html?issue=66&id=4#article
4. http://www.codethecode.com/projects/class-dump/
5. http://developer.apple.com/library/mac/#qa/qa1361/_index.html
6. http://landonf.bikemonkey.org/code/macosx/Leopard_PT_DENY_ATTACH.20080122.html
7. Mac OS X Internals, Amit Singh

References

8. Under the iHood, Recon 2008, Cameron Hotchkies.
9. http://dtrace.org/blogs/brendan/2011/10/10/top-10-dtrace-scripts-for-mac-os-x/
10. https://github.com/rentzsch/jrswizzle
11. http://radare.org/
12. http://www.hopperapp.com
13. https://github.com/gdbinit/pydbg64
14. http://lldb.llvm.org/
15. http://code.google.com/p/distorm/

References

16. http://otx.osxninja.com/
17. http://sourceforge.net/projects/machoview/
18. http://www.suavetech.com/0xed/0xed.html
19. http://cocoadev.com/wiki/MethodSwizzling
20. http://osxbook.com/book/bonus/ancient/whatismacosx/arch_xnu.html
21. http://chaosradio.ccc.de/24c3_m4v_2303.html
22. http://developer.apple.com/library/mac/technotes/tn2206/_index.html#//apple_ref/doc/uid/DTS40007919-CH1-SECTION7

Greets to:

snare, noar, saure, #osxre, 0d, put.as team

http://reverse.put.as

http://github.com/gdbinit

reverser@put.as

@osxreverser

#osxre @ irc.freenode.net