Encryption Wrapper
on OSX
Overview

- OSX Kernel
- What is an Encryption Wrapper?
- Project Implementation
OSX’s Origins and Design
NeXTSTEP Legacy

Jobs creates NeXT

1985

NextStep v1

1989

NextStep v3.3

1995

Apple acquisition

PPC Rhapsody

2001

Mac OS X

2005

Mac OS X i386
Adv. NetInfo.rtf
Path: /home/paul

Link To:
Size: 30.2KB
Owner: paul
Group: Wheel

Permissions
Read: √ √ √ √
Write: √ √ √ √
Owner: group: other

Changed: 6:20 PM SAT 22 NOV 1995

Attributes Inspector
Adv. NetInfo.rtf
AppMan diagram?
Apps
A SoftPC.local ...
Barclays
Bargains Props...
calendar
CISMail
CISNews
cmos.ram

Mailboxes
bbs post
Create
Dataphile
eof
MagicCap
MagicDev
netinfo-talk
next-droom
next-admin
next-classroom
next-icon
next-jobs
next-managers
next-PPP
next-prog
nfs-remote
nfs-remote
nfs-remote
nfs-remote
nfs-remote
nfs-remote

Name: CmmnWeb.rbox
Delete New
Open Transfer
NeXTSTEP Legacy

- Operating System Design
- Object Oriented (APIs, KPIs)
- OpenStep API (now cocoa)
- Objective-C
- Finder :-(
NeXTSTEP Design

Applications

- NeXT GUI / OpenStep API
- Display PostScript
- Loadable Kernel Servers
- DriverKit

4.3BSD

Mach 2.0/2.5

* Berkeley - Carnegie Mellon Universities
CMU’s Mach Original Goals

- “An operating system to create operating systems”
- Designed to replace the huge complex buggy Unix kernel
- Promoted portability, stability and simplicity
Mach Original Concept: a *MicroKernel*

- **Applications**
- **Real Operating System**
- **Mach Operating System**

User

Kernel
Mach, a MicroKernel
Mach Basics

- **Task**: simple resources container (VM, ports, threads,...)
- **Thread**: basic execution unit (shares task’s resources with other threads, only owns an execution context)
- **Port**: message queue protected with privileges
- **Message**: data that a thread can send or receive
- **Memory object**: container for data (mapped in the task’s virtual address space)
Mach Basics...

- **No** user/group/world notions
- **No** FS notions like working directory

- Pure/simple/secure oriented object concept
  - Only one object can own/receive messages on a specific port
  - A thread needs to have the right on a specific port to send a message to it
Mach Memory Management

Task A

Task B

RAM

VGA

I/O

I/O

I/O
funcA("hello", 1);

void funcA(char* s, int i) {
    // mig generated code
    msg.s = s;
    msg.i = i;
    mach_msg(port, msg);
}

// mig generated code
mach_msg(port, msg);
funcA(msg.s, msg.c);

void funcA(char* s, int i) {
    ...
}
Mach is responsible for...

- Preemptive multitasking (schedule user/kernel threads)
- Interrupt Management
- Protected Memory
- Virtual Memory Management
- IPC
- Real-Time Support
- Debugging support
- Console I/O
From CMU to Apple Implementation
CMU’s implementations

- 4.3BSD
- Mach 1.0-2.5
- Mach 3.0

User APIs (libc, ...)

Kernel

Applications
Monolithic system calls (FreeBSD, Linux, ...)

write

SYS_write()

user -> system switch

system -> user switch

write()
OSX: Colocation

- User
  - APIs (libc, ...)
  - Applications

- Kernel
  - FreeBSD
  - IOKit
  - Mach 3.0
BSD on top of Mach

- How to wrap a secure and simple micro-kernel into an insecure and complex monolithic-kernel?
BSD on top of Mach

Applications

APIs (libc, ...)
IOKit

- Driver framework
- Fully object Oriented
- C++ (w/o exceptions, templates and rtti)
- Enable to easily reuse code
static int hello_init(void) {
    return initlauncher();
}
static void hello_exit(void) {
}
module_init(hello_init);
module_exit(hello_exit);

static int hello_init(void) {
    #ifdef v2
        initnewV2stuff();
    #endif
    return initlauncher();
}
static void hello_exit(void) {
}
module_init(hello_init);
module_exit(hello_exit);
bool rocketl::init()
{
    return initlauncher();
}

class rocketlV2 : public rocketl
{
    
    bool rocketlV2::init()
    {
        if (initnewV2stuff())
        {
            return rocketl::init();
        }
        return FALSE;
    }
}
IOKit

- Meta Class
- Reference Counting
- Inheritance
- WorkLoop (tasklets)
IOKit’s author
Binary Formats

- a.out
- coff
- PE
- ELF
- Mach-O
Mach-O

- Multi-Archi
- Kernel is a Mach-O (kernel is always 32bits)

$ file /mach_kernel
/mach_kernel: Mach-O universal binary with 2 architectures
/mach_kernel (for architecture i386): Mach-O executable i386
/mach_kernel (for architecture ppc): Mach-O executable ppc
Mach-O

- LC_UUID
- LC_SEGMENT / LC_SEGMENT_64
- LC_SYMTAB
- LC_DSYMTAB
- LC_THREAD
- ....

Header

Load commands
- Load command 1
- Load command n

Data
- LC_SEGMENT __TEXT
- Section __text
- Section __cstring

LC_THREAD
- eax 0x0 ebx 0x0 ecx 0x0 edx 0x0
- edi 0x0 esi 0x0 ebp 0x0
- eip 0x00001f70
Design Considerations

- BSD designed by the academic world for the academic world
- Mach designed by the academic world for BSD
- NeXTSTEP designed for the education world
- OSX designed for???
Encryption Wrapper
What is binary encryption?

- Encrypt the binary image
- Decrypt the code at runtime
- Delay as much as possible the code decryption
- Guarantee that no single snapshot contains the whole unencrypted program
Threats

- Reverse engineers
- Crackers
- Forensic investigators
Why encrypt?

- Adding an anti piracy protection
  - Prevent unauthorized user from executing it
- Adding a reverse engineering protection
  - Prevent static attacks (binary static analysis)
- Hindering forensic analysis, just in case...
What is binary encryption?

(__TEXT,__text) section
start:
00001f70	 pushl	 $0x00
00001f72	 movl	 %esp,%ebp
00001f74	 andl	 $0xf0,%esp
00001f77	 subl	 $0x10,%esp

(__TEXT,__text) section
start:
8>^KB^@^@<89>^^P<FF><85>
$<FF><FF><FF><8B><85><F6>^O<85>
><91><FC><FF><FF><8B><9D>$
<FF><FF><FF><85><DB>^O<84>{^A^@
@^@<8B><85>@<FF><FF><FF>^O

Section 1 data “__text”

Segment __TEXT

Load commands

Segment command 1

Segment command n

Header

Section 1 data “__cstring”

Segment __TEXT

Load commands

Segment command 1

Segment command n

Encryption

Header

Section 1 data “__cstring”
Page Fault

Page Fault
- 0x1feb: pushl %ebp
- 0x1fec: movl %esp,%ebp
- 0x1fee: subl $0x08,%esp
- 0x1ff1: calll _hello (0x2000)
- 0x1ff6: leave
- 0x1ff7: ret

_hello:
- 0x2000: pushl %ebp

Task A

hello.exe
What didn’t work...

- Patch the Kernel Source
- Add a custom Pager
- Fake Device
Patch the Kernel Source

- Too invasive... No one on OSX installs other kernels than the ones released by Apple
- Really hard to recompile a custom kernel, no support
- Using kernel module is really “mandatory” (cf. Apple)
What is a pager?

- Responsible for reading/writing pages to the backing store
- Abstraction layer between the Kernel and the backing store
OSX’s Pager

- Pager selection is hard coded (VNode pager, ...)
- No clean mechanism to add a custom pager
- The only way is by hooking OSX’s page fault handler (dirty)
Fake Device

- Can limit accesses to kernel only
- However, can’t guarantee that the client is really our driver
- Can’t guarantee the amount of unencrypted pages
What works?

- Exceptions Hooking

```
start:
00001f70  pushl $0x00
00001f72  movl %esp,%ebp
```

**Protection Exception**

**Page Fault**

- **Segment **__TEXT**

  - Section 1 data "__text"
  - (__TEXT,__text) section
  - VM_PROT_NONE

- **0x1000**
  - 8>&^KB^@^@<89>^^P<FF><85>
  - $<FF><FF><FF><8B><85><F

- **Kernel**

- **Pager**

- **Backing Store**
Exceptions Hooking

- Clear Mach API to override task exception handler
- No overhead (no need to call the original handler)
- Thanks to Mach the User-land and the Kernel-land are both clients of Mach. The same code works in both lands
Demo
Reference

- **Inside the Mac OS X Kernel** - *Lucy <whoislucy(at)gmail.com>*
- **Meet Mach** - *James Scott*
- **IOKit Fundamentals** - *Apple*
- **Phrack p58-10 “Binary Encryption on Unix”** - *grugq <grugq@lokmail.net>*
- **The NeXT Chapter** - *kernelthread.com*
- **Infecting the Mach-o Object Format** - *Neil Archibald*
- **MACH Kernel Interface Manual** - *CMU*