Under the iHood

DEFCON 16

Cameron Hotchkies [1]

[1] chotchkies@tippingpoint.com

DEFCON 16

About Me

- Work at TippingPoint's Digital Vaccine Labs
  - Responsible for vuln-discovery, patch analysis, product security
  - Keep tabs on us at http://dvlabs.tippingpoint.com
- Author and contributor to:
  - PaiMei Reverse Engineering Framework
  - Absinthe SQL Injection tool
- Side projects:
  - XSO - OS X Reversers: http://0x90.org/mailman/listinfo/xso

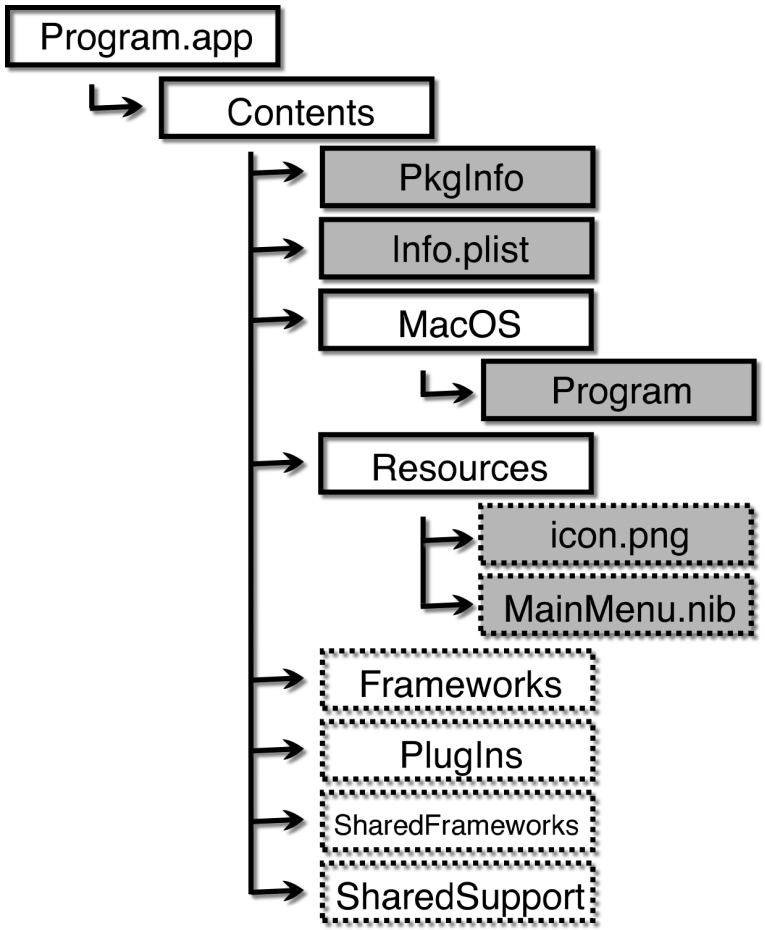
Talk Outline

- File Formats
- Tools
- Common Disassembly Patterns
- Carbon
- Objective-C
- Case Study (Mac vs. Windows)

Applications

- Applications in OS X are stored in a directory structure referred to as *bundle*s or *package*s
- Finder will treat any directory ending in .app as a single entity
- self contained package with the binary and all necessary resources

Application directory structure



Info.plist

- XML or binary based list of application properties
- contains data such as major & minor version numbers, icon names, etc
- Well documented by Apple
- use *plutil* to convert between xml and binary formats
- "The plutil command obeys no one's rules but its own."

PkgInfo

- APPL indicates an apple application bundle
- No relevant information in the file
- 4-byte package type followed by the 4-byte signature

Mach-O

- the standard binary format on OS X
- identified by the magic number 0xFEEDFACE
- 0xFEEDFACF on 64-bit
- Fat/Universal binaries include code for multiple architectures
- Fat binaries are identified by 0xCAFEBABE

Mach-O

- the standard binary format on OS X
- identified by the magic number 0xFEEDFACE
- 0xFEEDFACF on 64-bit
- Fat/Universal binaries include code for multiple architectures
- Fat binaries are identified by 0xCAFEBABE
- yes, this is the same as Java

Mach-O

- the standard binary format on OS X
- identified by the magic number 0xFEEDFACE
- 0xFEEDFACF on 64-bit
- Fat/Universal binaries include code for multiple architectures
- Fat binaries are identified by 0xCAFEBABE
- yes, this is the same as Java
- Googling "mach-o" is a fun game on it's own
- "Can black-hole MACHO binaries be detected by the Brazilian spherical antenna?"

Mach-O Text Segment

.text	(__TEXT,__text)	Code, same as everywhere else
.const	(__TEXT,__const)	Initialized constants
.static_const	(__TEXT,__static_const)	Not defined*
.cstring	(__TEXT,__cstring)	Null terminated byte strings
.literal4	(__TEXT,__literal4)	4 byte literals
.literal8	(__TEXT,__literal8)	8 byte literals
.constructor	(__TEXT,__constructor)	C++ constructors*
.destructor	(__TEXT,__destructor)	C++ destructors*
.fvmlib_init0	(__TEXT,__fvmlib_init0)	fixed virtual memory shared library initialization*
.fvmlib_init1	(__TEXT,__fvmlib_init1)	fixed virtual memory shared library initialization*
.symbol_stub	(__TEXT,__symbol_stub)	Indirect symbol stubs
.picsymbol_stub	(__TEXT,__picsymbol_stub)	Position-independent indirect symbol stubs.
.mod_init_func	(__TEXT,__mod_init_func)	C++ constructor pointers*

Mach-O Data Segment

.data	(__DATA,__data)	Initialized variables
.static_data	(__DATA,__static_data)	Unused*
.non_lazy_symbol_pointer	(__DATA,__nl_symbol_pointer)	Non-lazy symbol pointers
.lazy_symbol_pointer	(__DATA,__la_symbol_pointer)	Lazy symbol pointers
.dyld	(__DATA,__dyld)	Placeholder for dynamic linker
.const	(__DATA,__const	Initialized relocatable constant variables
.mod_init_func	(__DATA,__mod_init_func)	C++ constructor pointers
.mod_term_func	(__DATA,__mod_term_func)	Module termination functions.
.bss	(__DATA,__bss)	Data for uninitialized static variables
.common	(__DATA,__common)	Uninitialized imported symbol definitions

Objective-C Segment

.objc_class	(__OBJC,__class)
.objc_meta_class	(__OBJC,__meta_class)
.objc_cat_cls_meth	(__OBJC,__cat_cls_meth)
.objc_cat_inst_meth	(__OBJC,__cat_inst_meth)
.objc_protocol	(__OBJC,__protocol)
.objc_string_object	(__OBJC,__string_object)
.objc_cls_meth	(__OBJC,__cls_meth)
.objc_inst_meth	(__OBJC,__inst_meth)
.objc_cls_refs	(__OBJC,__cls_refs)
.objc_message_refs	(__OBJC,__message_refs)
.objc_symbols	(__OBJC,__symbols)
.objc_category	(__OBJC,__category)
.objc_class_vars	(__OBJC,__class_vars)
.objc_instance_vars	(__OBJC,__instance_vars)
.objc_module_info	(__OBJC,__module_info)
.objc_class_names	(__OBJC,__class_names)
.objc_meth_var_names	(__OBJC,__meth_var_names)
.objc_meth_var_types	(__OBJC,__meth_var_types)
.objc_selector_strs	(__OBJC,__selector_strs)

Objective-C Segment

.objc_class	(__OBJC,__class)
.objc_meta_class	(__OBJC,__meta_class)
.objc_cat_cls_meth	(__OBJC,__cat_cls_meth)
.objc_cat_inst_meth	(__OBJC,__cat_inst_meth)
.objc_protocol	(__OBJC,__protocol)
.objc_string_object	(__OBJC,__string_object)
.objc_cls_meth	(__OBJC,__cls_meth)
.objc_inst_meth	(__OBJC,__inst_meth)
.objc_cls_refs	(__OBJC,__cls_refs)
.objc_message_refs	(__OBJC,__message_refs)
.objc_symbols	(__OBJC,__symbols)
.objc_category	(__OBJC,__category)
.objc_class_vars	(__OBJC,__class_vars)
.objc_instance_vars	(__OBJC,__instance_vars)
.objc_module_info	(__OBJC,__module_info)
.objc_class_names	(__OBJC,__class_names)
.objc_meth_var_names	(__OBJC,__meth_var_names)
.objc_meth_var_types	(__OBJC,__meth_var_types)
.objc_selector_strs	(__OBJC,__selector_strs)

What they say: "All sections in the __OBJC segment, including old sections that are no longer used and future sections that may be added, are exclusively reserved for the Objective C compiler's use."

Objective-C Segment

.objc_class	(__OBJC,__class)
.objc_meta_class	(__OBJC,__meta_class)
.objc_cat_cls_meth	(__OBJC,__cat_cls_meth)
.objc_cat_inst_meth	(__OBJC,__cat_inst_meth)
.objc_protocol	(__OBJC,__protocol)
.objc_string_object	(__OBJC,__string_object)
.objc_cls_meth	(__OBJC,__cls_meth)
.objc_inst_meth	(__OBJC,__inst_meth)
.objc_cls_refs	(__OBJC,__cls_refs)
.objc_message_refs	(__OBJC,__message_refs)
.objc_symbols	(__OBJC,__symbols)
.objc_category	(__OBJC,__category)
.objc_class_vars	(__OBJC,__class_vars)
.objc_instance_vars	(__OBJC,__instance_vars)
.objc_module_info	(__OBJC,__module_info)
.objc_class_names	(__OBJC,__class_names)
.objc_meth_var_names	(__OBJC,__meth_var_names)
.objc_meth_var_types	(__OBJC,__meth_var_types)
.objc_selector_strs	(__OBJC,__selector_strs)

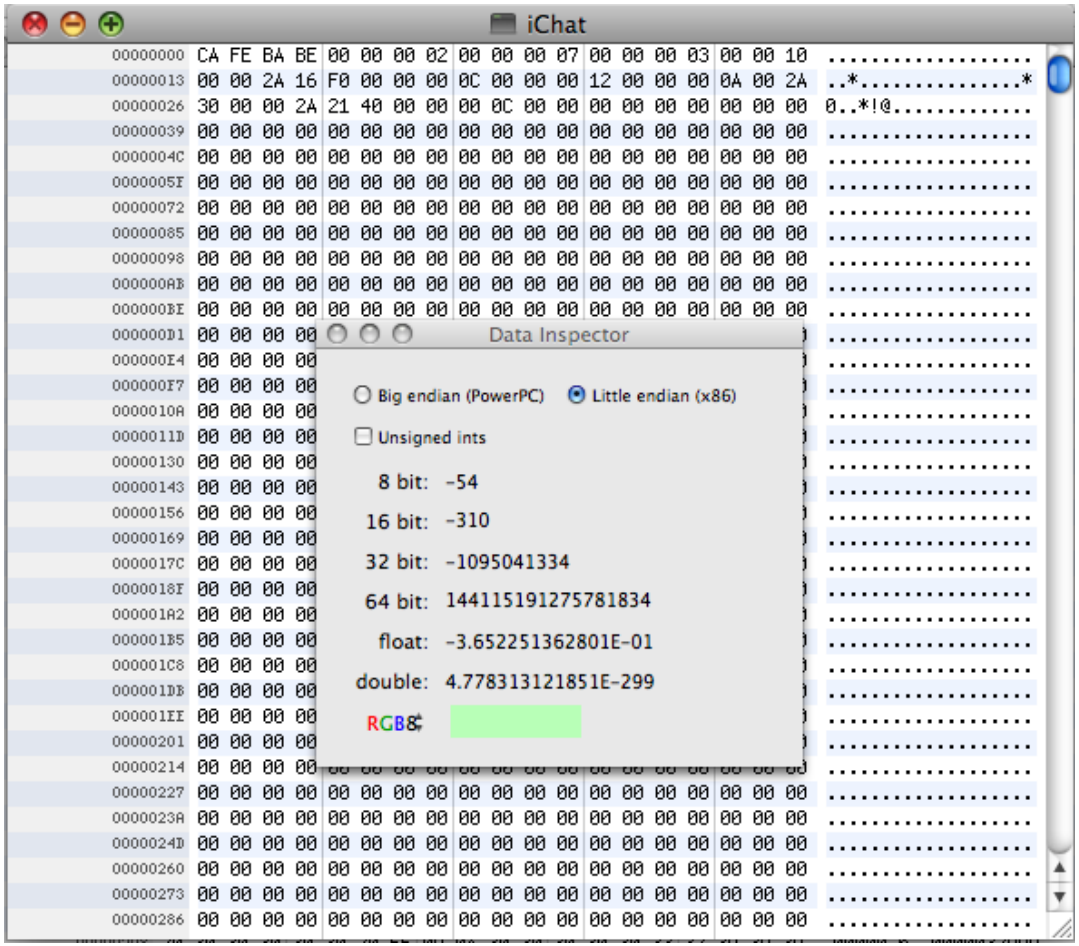
What they say: "All sections in the __OBJC segment, including old sections that are no longer used and future sections that may be added, are exclusively reserved for the Objective C compiler's use."
What they mean: "No docs 4 u LOL kthxbai!"

vmmap

- available standard on OS X
- lists memory mapping for a binary at runtime
- includes segment partitions
- quick way to track down what address is heap/stack/library without a debugger

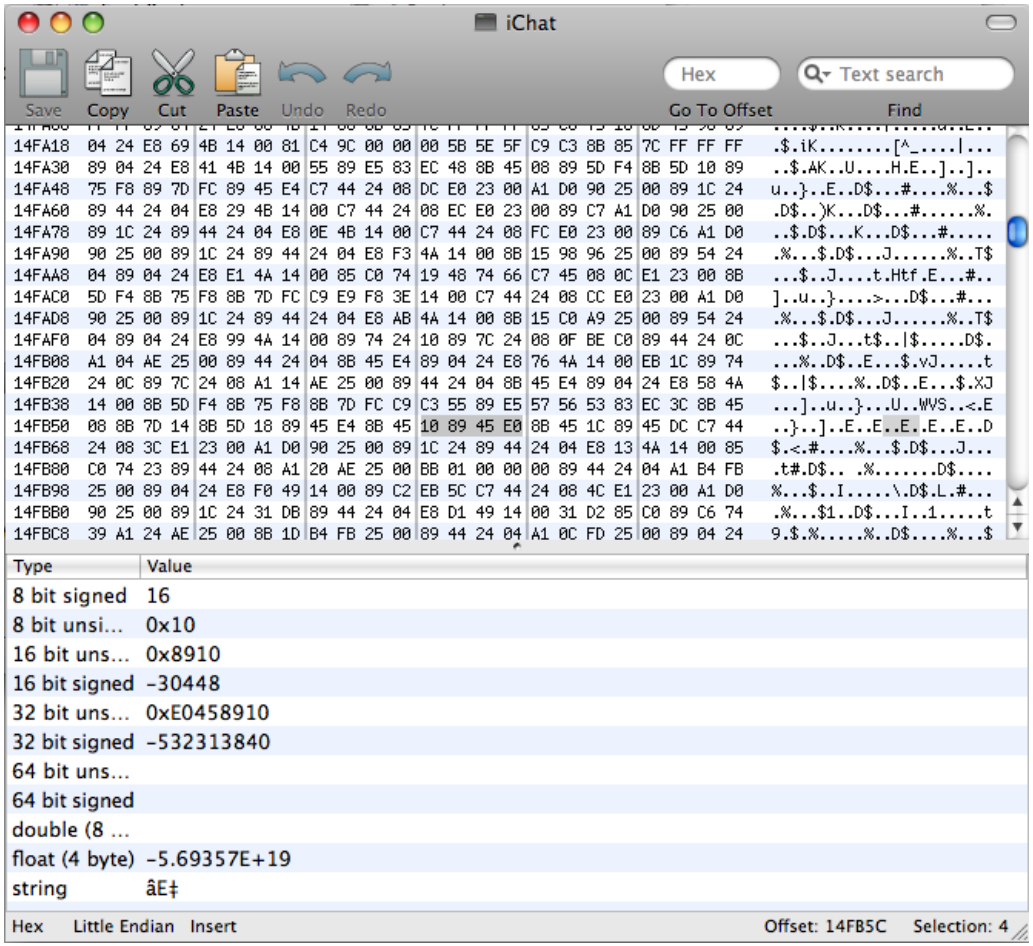
Hex Fiend

- An open source hex editor, that is not very difficult to modify.
- http://ridiculousfish.com/hexfiend/



0xED

- Another hex editor, has plugins to display/edit custom data types.
- http://www.suavetech.com/0xed/0xed.html



otool

- the mac equivalent of objdump, available in a default install.
- use 'otool -otV' to resolve symbols

```
Camtronic-2:MacOS cameron$ otool -otV iChat | less
iChat:
(__TEXT,__text) section
00003c38        pushl   $0x00
00003c3a        movl    %esp,%ebp
00003c3c        andl    $0xf0,%esp
00003c3f        subl    $0x10,%esp
00003c42        movl    0x04(%ebp),%ebx
00003c45        movl    %ebx,0x00(%esp)
00003c49        leal    0x08(%ebp),%ecx
00003c4c        movl    %ecx,0x04(%esp)
00003c50        addl    $0x01,%ebx
00003c53        shll    $0x02,%ebx
00003c56        addl    %ecx,%ebx
00003c58        movl    %ebx,0x08(%esp)
00003c5c        movl    (%ebx),%eax
00003c5e        addl    $0x04,%ebx
00003c61        testl   %eax,%eax
00003c63        jne     0x00003c5c
00003c65        movl    %ebx,0x0c(%esp)
00003c69        calll   0x0013be3d
00003c6e        movl    %eax,0x00(%esp)
00003c72        calll   0x002944f7      ; symbol stub for: _exit
00003c77        hlt
```

otool

- use 'otool -L' to list required libraries

```
Camtronic-2:MacOS cameron$ otool -L iChat
iChat:
/System/Library/Frameworks/IOBluetooth.framework/Versions/A/IOBluetooth
(compatibility version 1.0.0, current version 1.0.0)
/System/Library/PrivateFrameworks/DisplayServices.framework/Versions/A/DisplayServices
(compatibility version 1.0.0, current version 1.0.0)
/System/Library/Frameworks/Cocoa.framework/Versions/A/Cocoa
(compatibility version 1.0.0, current version 12.0.0)
/System/Library/Frameworks/ApplicationServices.framework/Versions/A/ApplicationServices
(compatibility version 1.0.0, current version 34.0.0)
/System/Library/Frameworks/AddressBook.framework/Versions/A/AddressBook
(compatibility version 1.0.0, current version 688.0.0)
/System/Library/Frameworks/InstantMessage.framework/Versions/A/InstantMessage
(compatibility version 1.0.0, current version 579.0.0)
/System/Library/Frameworks/QuickTime.framework/Versions/A/QuickTime
(compatibility version 1.0.0, current version 14.0.0)
/System/Library/PrivateFrameworks/VideoConference.framework/Versions/A/VideoConference
(compatibility version 2.0.0, current version 2.0.0)
```

otx

- A tool used to clean up output from otool
- http://otx.osxninja.com/

```
+56    00003cda    a3d8c42400      movl    %eax,0x0024c4d8
+61    00003cdf    a1d4902500      movl    0x002590d4,%eax     alloc
+66    00003ce4    89442404        movl    %eax,0x04(%esp)
+70    00003ce8    a1b0fb2500      movl    0x0025fbb0,%eax     NSMutableArray
+75    00003ced    890424          movl    %eax,(%esp)
+78    00003cf0    e89d082900      calll   0x00294592          +[NSMutableArray alloc]
+83    00003cf5    8b1570912500    movl    0x00259170,%edx     init
+89    00003cfb    89542404        movl    %edx,0x04(%esp)
+93    00003cff    890424          movl    %eax,(%esp)
+96    00003d02    e88b082900      calll   0x00294592          -[(%esp,1) init]
```

classdump

- Similar to "otool -ov" but represents code as Objective C declarations.
- http://www.codethecode.com/projects/class-dump/

```
Camtronic-2:MacOS cameron$ class-dump iChat
/*
* Generated by class-dump 3.1.2.
*
* class-dump is Copyright (C) 1997-1998, 2000-2001, 2004-2007 by Steve Nygard.
*/
...

@interface SmileyCell : NSButtonCell
{
  NSString *_axDescription;
}
- (void)dealloc;
- (id)accessibilityAttributeNames;
- (void)accessibilitySetValue:(id)fp8 forAttribute:(id)fp12;
- (id)accessibilityAttributeValue:(id)fp8;
- (void)drawInteriorWithFrame:(struct _NSRect)fp8 inView:(id)fp24;
@end
```

IDA Pro

- IDA Pro for windows works fine with Parallels
- IDA Pro for OS X runs on the console
- http://hex-rays.com/idapro/
- http://www.parallels.com/
- Both IDA & Parallels are commercial (not-free)

Debuggers

- Charlie Miller ported pyDBG to OSX
- Stock installs come with gdb
- pygdb available at http://code.google.com/p/pygdb/
- vtrace at https://www.kenshoto.com/vtrace/
- Weston & Beauchamp will also be releasing reDBG soon, a ruby debugger.

RE:Trace

- Introduced at Black Hat DC 2008. RE:Trace is a Ruby framework to interact with dtrace
- http://www.poppopret.org/

Calling Conventions

On OS X, std_call is the calling convention. As it is compiled with GCC, stack space is allocated at the function start.
Variables are moved in, not pushed onto the stack

```
__text:00001D01    call    $+5
__text:00001D06    pop     ebx
__text:00001D07    lea     eax, [ebx+1316h]
__text:00001D0D    mov     eax, [eax]      ; "NSAutoreleasePool"
__text:00001D0F    mov     edx, eax
__text:00001D11    lea     eax, [ebx+130Eh]
__text:00001D17    mov     eax, [eax]      ; "alloc"
__text:00001D19    mov     [esp+98h+var_94], eax
__text:00001D1D    mov     [esp+98h+var_98], edx
__text:00001D20    call    _objc_msgSend
__text:00001D25    mov     edx, eax
__text:00001D27    lea     eax, [ebx+130Ah] ; "init"
__text:00001D2D    mov     eax, [eax]
__text:00001D2F    mov     [esp+98h+var_94], eax
__text:00001D33    mov     [esp+98h+var_98], edx
__text:00001D36    call    _objc_msgSend
```

Calling Conventions

On OS X, std_call is the calling convention. As it is compiled with GCC, stack space is allocated at the function start.
Variables are moved in, not pushed onto the stack

```
__text:00001D01    call    $+5
__text:00001D06    pop     ebx
__text:00001D07    lea     eax, [ebx+1316h]
__text:00001D0D    mov     eax, [eax]      ; "NSAutoreleasePool"
__text:00001D0F    mov     edx, eax
__text:00001D11    lea     eax, [ebx+130Eh]
__text:00001D17    mov     eax, [eax]      ; "alloc"
__text:00001D19    mov     [esp+98h+var_94], eax
__text:00001D1D    mov     [esp+98h+var_98], edx
__text:00001D20    call    _objc_msgSend
__text:00001D25    mov     edx, eax
__text:00001D27    lea     eax, [ebx+130Ah] ; "init"
__text:00001D2D    mov     eax, [eax]
__text:00001D2F    mov     [esp+98h+var_94], eax
__text:00001D33    mov     [esp+98h+var_98], edx
__text:00001D36    call    _objc_msgSend
```

before renaming variables, check the stack delta

Local Addressing

Functions will frequently refer to an address that is not the base of the function, just an anchor point
This is used frequently in data references and jump tables

```
__text:000E63CF mov eax, [ebx+eax*4+300h]
__text:000E63D6 add eax, ebx
__text:000E63D8 jmp eax
__text:000E63D8 ;
__text:000E63DA align 4 ; Jump table
__text:000E63DC dd 2 dup( 0A80h), 7AAh, 7B9h, 3A4h, 716h,3 dup( 0A80h), 94Ch, 9E0h
__text:000E63DC dd 3FAh, 0A80h, 0A24h,4 dup( 0A80h), 998h,2 dup( 0A80h), 435h, 7C8h
__text:000E63DC dd 3 dup( 7E7h),0Ch dup( 0A80h), 7F6h, 0A80h, 905h, 6AF48D8Bh, 758BFFFBh, 8418B08h
```

Anchor Function

This function is used to generate a local anchor

```
get_pc proc near
  mov ebx, [esp+0]
  retn
get_pc endp
```

Anchor Function

This function is used to generate a local anchor

```
get_pc proc near
  mov ebx, [esp+0]
  retn
get_pc endp
```

Or it can be inlined:

```
call $+5
pop ebx
```

Anchor to data

```
__text:00001D01    call    $+5
__text:00001D06    pop     ebx
__text:00001D07    lea     eax, [ebx+1316h]
__text:00001D0D    mov     eax, [eax]      ; "NSAutoreleasePool"
__text:00001D0F    mov     edx, eax
__text:00001D11    lea     eax, [ebx+130Eh]
__text:00001D17    mov     eax, [eax]      ; "alloc"
__text:00001D19    mov     [esp+98h+var_94], eax
__text:00001D1D    mov     [esp+98h+var_98], edx
__text:00001D20    call    _objc_msgSend
__text:00001D25    mov     edx, eax
__text:00001D27    lea     eax, [ebx+130Ah] ; "init"
__text:00001D2D    mov     eax, [eax]
__text:00001D2F    mov     [esp+98h+var_94], eax
__text:00001D33    mov     [esp+98h+var_98], edx
__text:00001D36    call    _objc_msgSend
```

Carbon

- Carbon is the 32-bit framework for interacting with the OS X system libraries.
- descended from the original Mac Toolbox
- Apple encourages it to be used as a stepping stone to Cocoa/Objective-C
- HI, CG

Objective-C

- Created in the mid 1980s by Stepstone
- Popularized by NeXT in the late 1980s
- Object Oriented inspired by Smalltalk
- Small set of decorators on top of C
- Functions aren't called, messages are sent
- Unicode strings are the standard, but stored internally as null terminated UTF8 strings
- Libraries are referred to as Frameworks

Frameworks

- Objective-C has a rich set of base framework classes to call from
- Common framework classes are prepended with NS (NeXTStep) or CF (Core Foundation)
- Other frameworks also make use of a two capital prefix
- NS is frequently a wrapper for CF (*toll-free bridge*), NSString == CFString
- The Objective-C system API for OS X is named Cocoa

AppKit

- AppKit is the GUI framework classes available for Cocoa
- iPhone uses UIKit instead, a scaled down version with some custom libraries.
- AppKit uses NS prefix, UIKit uses UI

Objective-C Methods

```
x = [object statement:arg1 second:arg2];
```

Objective-C Methods

```
x = [object statement:arg1 second:arg2];
```

Component:

Selector decorators

Objective-C Methods

```
x = [object statement:arg1 second:arg2];
```

Component:

Selector decorators

Component:

Recipient

Objective-C Methods

```
x = [object statement:arg1 second:arg2];
```

Component:

Selector decorators

Component:

Recipient

Component:

Selector

Objective-C Methods

```
x = [object statement:arg1 second:arg2];
```

Component:

Selector decorators

Component:

Recipient

Component:

Selector

Component:

Arguments

msgSend

Calls to selectors are just wrappers around C functions:

```
id objc_msgSend(object, "statement:second:", arg1, arg2);
```

msgSend

Calls to selectors are just wrappers around C functions:

```
id objc_msgSend(object, "statement:second:", arg1, arg2);
```

Component:

Recipient

Component:

Selector

Component:

Arguments

msgSendSuper

The objc_msgSendSuper() function works in the same way that obj_msgSend() does, but passes it to the superclass. The recipient in the call to the objc_msgSendSuper() is an objc_super data structure.

```
id objc_msgSendSuper(struct objc_super *super, SEL op, ...);
```

msgSend_fpret

The objc_msgSend_fpret() function is identical to the standard objc_msgSend() function, differing only in the fact that the return value is a floating point instead of an integral type.

```
double objc_msgSend_fpret(id self, SEL op, ...);
```

msgSend_stret

The objc_msgSend_stret() function is used to return a structure instead of a value. The first argument to the objc_msgSend_stret() function is a pointer to memory large enough to contain the returning structure.

```
void objc_msgSend_stret(void * stretAddr, id theReceiver,
SEL theSelector, ...);
```

msgSendSuper_stret

Send to the superclass, get a structure back.

```
void objc_msgSendSuper_stret(void * stretAddr, struct objc_su
*super, SEL op, ...);
```

msgSend in asm

Since Objective C uses message passing between selectors, this means there are no direct calls between functions.

```
mov     [esp+38h+var_30], eax
mov     eax, ds:off_400040
mov     [esp+38h+var_34], eax
mov     eax, ds:off_4000DC
mov     [esp+38h+var_38], eax
call    _objc_msgSend
mov     [esp+38h+var_28], 0
mov     [esp+38h+var_24], 404E0000h
mov     [esp+38h+var_2C], 0
mov     [esp+38h+var_30], eax
mov     eax, ds:off_40003C
mov     [esp+38h+var_38], esi
mov     [esp+38h+var_34], eax
call    _objc_msgSend
```

msgSend in asm

Determining the selectors automatically is not difficult.

```
mov     esi, ds:off_4000D8                      ; "NSURLRequest"

mov     [esp+38h+var_30], eax                   ; arg1
mov     eax, ds:off_400040                      ; "URLWithString:"
mov     [esp+38h+msgSend_selector], eax
mov     eax, ds:off_4000DC                      ; "NSURL"
mov     [esp+38h+msgSend_recipient], eax
call    _objc_msgSend                           ; a = [NSURL URLWithString:]

mov     [esp+38h+var_28], 0                     ; arg3
mov     [esp+38h+var_24], 404E0000h
mov     [esp+38h+var_2C], 0                     ; arg2
mov     [esp+38h+var_30], eax                   ; arg1 (retVal from [NSURLWithString])
mov     eax, ds:off_40003C                      ; "requestWithURL:cachePolicy:timeoutInterval:"
mov     [esp+38h+msgSend_recipient], esi
mov     [esp+38h+msgSend_selector], eax
call    _objc_msgSend                           ; a=[NSURLRequest requestWithURL:cachePolicy:timeoutInterval:]
```

Selector Structures in the Binary

All of the information for selectors are stored in the _OBJC segment of the binaries

```
__inst_meth:00400220 DownloadDelegate__mthd dd 0 ; DATA XREF: __class:DownloadDelegate
__inst_meth:00400224 dd 9
__inst_meth:00400228 dd offset aDownloadDidr_0, offset aV16@04@8i12, offset download_didReceiveDataOfLength_
; "download:didReceiveDataOfLength:"
__inst_meth:00400234 dd offset aDownloadDidrec, offset aV16@04@8@12, offset download_didReceiveResponse_
; "download:didReceiveResponse:"
__inst_meth:00400240 dd offset aDownloadDidcre, offset aV16@04@8@12, offset download_didCreateDestination_
; "download:didCreateDestination:"
```

Type Encodings

Table: Objective-C Type Encodings

Code	Description	Code	Description
c	char	C	unsigned char
s	short	S	unsigned short
i	int	I	unsigned int
l	long	L	unsigned long
q	long long	Q	unsigned long long
f	float	d	double
B	C++ bool / C99 _Bool	v	void
*	c_string (char *)	@	object / id
#	class (Class)	:	selector (SEL)
[array type]	array	{name=type}	structure
(name=type)	union	?	unknown / function pointer
^type	pointer to *type*	b*num*	bitfield of *num* bytes

Argument Type Encoding

In the method definition sections (as well as the ivars) the data types for each argument are described using type encodings.

```
v16@0:4@8@12
```

Argument Type Encoding

In the method definition sections (as well as the ivars) the data types for each argument are described using type encodings.

```
v16@0:4@8@12
-(void)
```

Argument Type Encoding

In the method definition sections (as well as the ivars) the data types for each argument are described using type encodings.

```
v16@0:4@8@12
-(void)
```

Argument Type Encoding

In the method definition sections (as well as the ivars) the data types for each argument are described using type encodings.

```
v16@0:4@8@12
-(void)method:
```

Argument Type Encoding

In the method definition sections (as well as the ivars) the data types for each argument are described using type encodings.

```
v16@0:4@8@12
-(void)method:(id) object1
```

Argument Type Encoding

In the method definition sections (as well as the ivars) the data types for each argument are described using type encodings.

```
v16@0:4@8@12
-(void)method:(id) object1 andthen:(id) object2
```

Argument Type Encoding

In the method definition sections (as well as the ivars) the data types for each argument are described using type encodings.

```
v16@0:4@8@12
-(void)method:(id) object1 andthen:(id) object2
```

Stack offsets are indicated and can be used to determine variable size when not implicitly defined

Standard Memory Management

Objective C uses reference counting to control memory allocations.

```
// Allocate memory
NSObject *object = [[NSObject alloc] init];
```

Standard Memory Management

Objective C uses reference counting to control memory allocations.

```
// Allocate memory
NSObject *object = [[NSObject alloc] init];

// removes the local reference
[object release];
```

Standard Memory Management

Objective C uses reference counting to control memory allocations.

```
// Allocate memory
NSObject *object = [[NSObject alloc] init];

// removes the local reference
[object release];

// adds a local reference to keep external objects in scope
[otherObject retain];
```

Autorelease Pools

To dispose of memory allocated by child functions, Objective C utilizes an object called an Autorelease Pool.

```
-(void) someFunction
{
  NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

  ...

  [pool release];
  return;
}
```

pools can be nested within loops, so expect to see multiple instances in larger functions

Garbage Collection

- Garbage collection was added in OS X 10.5
- Classes designed for GC can be identified by having a finalize selector
- can be triggered by the *collectExhaustively* and *collectIfNeeded* selectors for NSGarbageCollector
- Garbage collection is *not* available on the iPhone, so you shouldn't see it there

Categories

- Categories are the ability to add functionality to a class from an external source
- This allows base foundation classes to be overridden
- If there's a category for any base class method signature, you need to rethink assumptions on code behaviour
- Category definitions are in the obviously labelled `cat_` sections of the binary

Timers

- Commonly used in protection schemes
- Objective-C supports multiple ways to create a timer
- NSTimer or NSOperationQueue

Windows

- Windows applications have had decades of people advancing cracking/packing
- Lots of documentation, but lots of hurdles

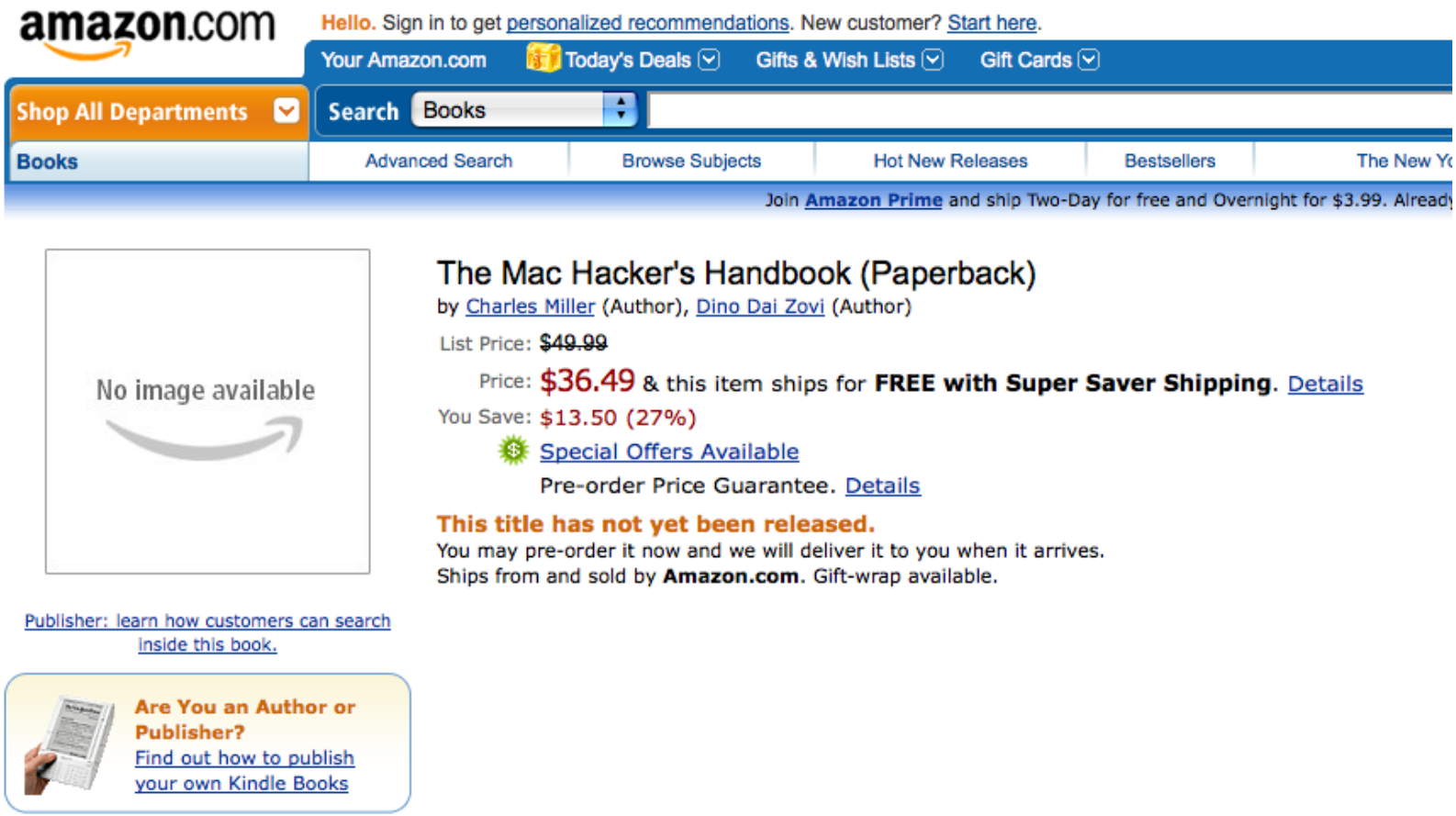
Mac

- Compressed/packed executables are not commonplace on OS X
- This section's slides exists only on the CD and are a temporal figment of your imagination
- That means you should probably have either gone to my presentation
- or at least get the full version off my website

References:

http://felinemenace.org/ nemo/
itsme's objc fixer:
http://nah6.com/ itsme/cvs-xdadevtools/ida/idcscripts/fixobjc.idc
fileoffset's otx parser
http://fileoffset.blogspot.com/2008/02/lua-script.html
http://www.dribin.org/dave/blog/archives/2006/04/22/tracing_objc/
http://unixjunkie.blogspot.com/

Coming Soon:


amazon.com
Hello. Sign in to get personalized recommendations. New customer? Start here.
Your Amazon.com
Today's Deals
Gifts & Wish Lists
Gift Cards
Shop All Departments
Search
Books
Books
Advanced Search
Browse Subjects
Hot New Releases
Bestsellers
Join Amazon Prime and ship Two-Day for free and Overnight for $3.99. Already
No image available
The Mac Hacker's Handbook (Paperback)
by Charles Miller (Author), Dino Dai Zovi (Author)
List Price: $49.99
Price: $36.49 & this item ships for FREE with Super Saver Shipping. Details
You Save: $13.50 (27%)
Special Offers Available
Pre-order Price Guarantee. Details
This title has not yet been released.
You may pre-order it now and we will deliver it to you when it arrives.
Ships from and sold by Amazon.com. Gift-wrap available.
Publisher: learn how customers can search inside this book.
Are You an Author or Publisher?
Find out how to publish your own Kindle Books

Total Slide Count

50