Low Level Stuff

- https://github.com/DerekSelander/lldb

Schedule

(time permitting)

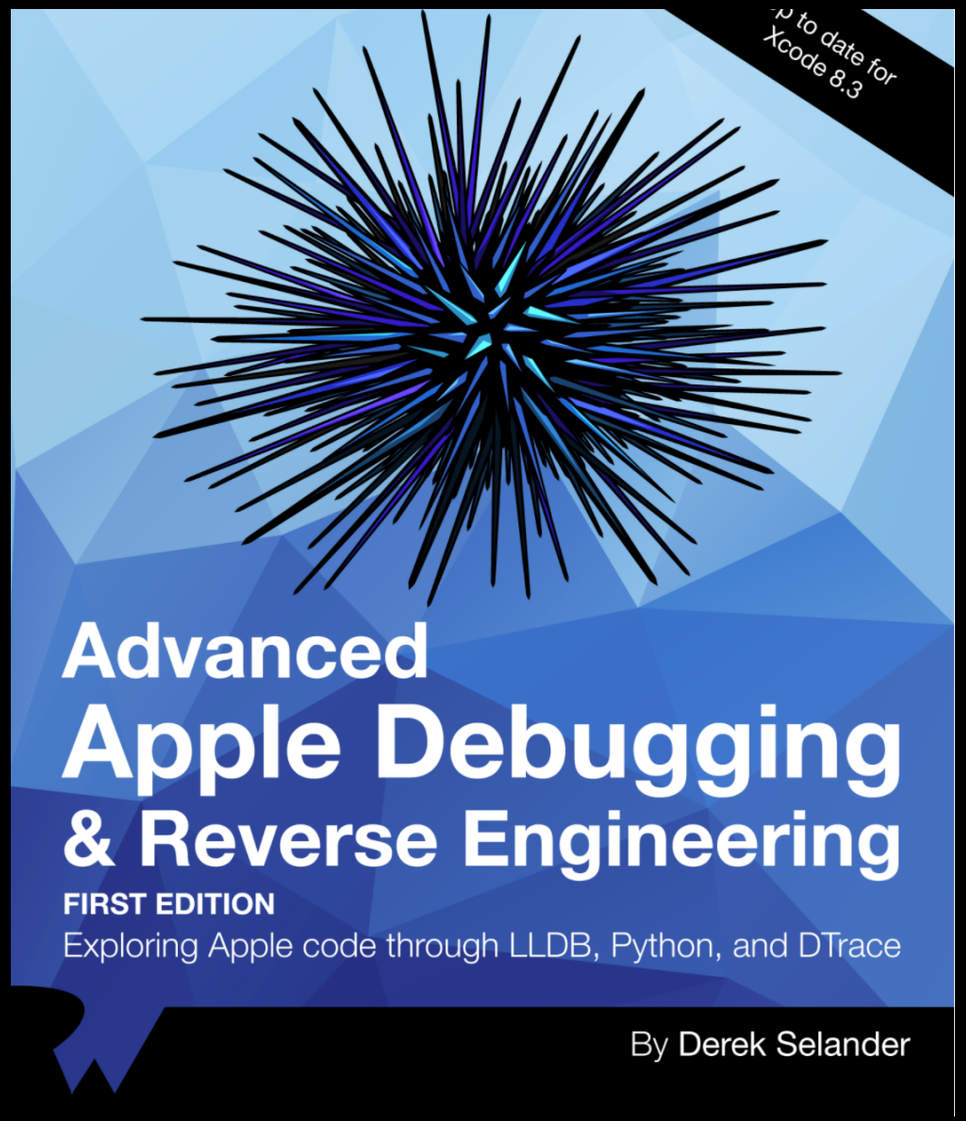
- x86_64 101: Calling conventions
- How I debug stuff (Demo w/ my LLDB tools)
- Mach-O 101: Segments, Sections & Load Commands
- Function Interposition (workshop)

The Tools

- https://github.com/DerekSelander/lldb

The Tools

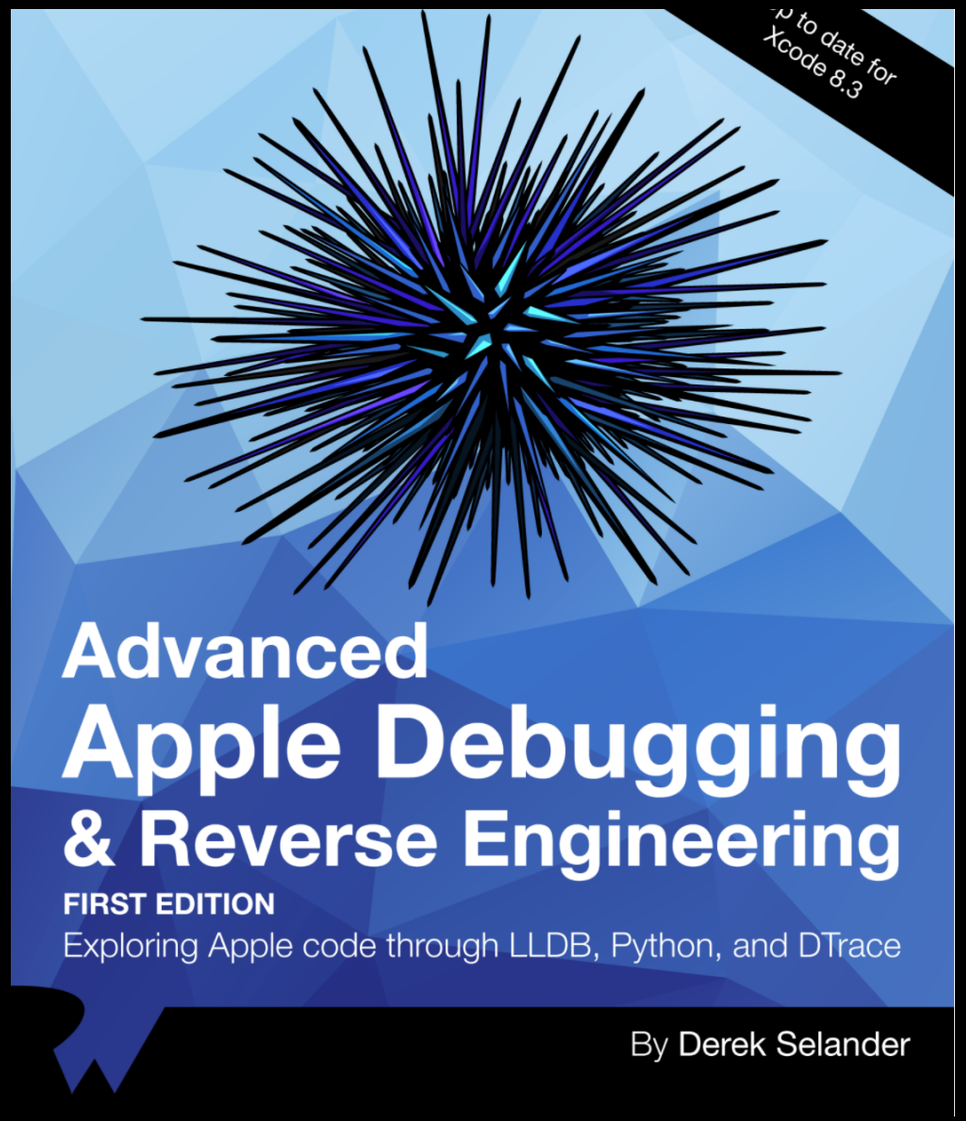
- https://github.com/DerekSelander/lldb



The Tools

- https://github.com/DerekSelander/lldb

- **search** - enumerates the heap to find objects of a certain type of class

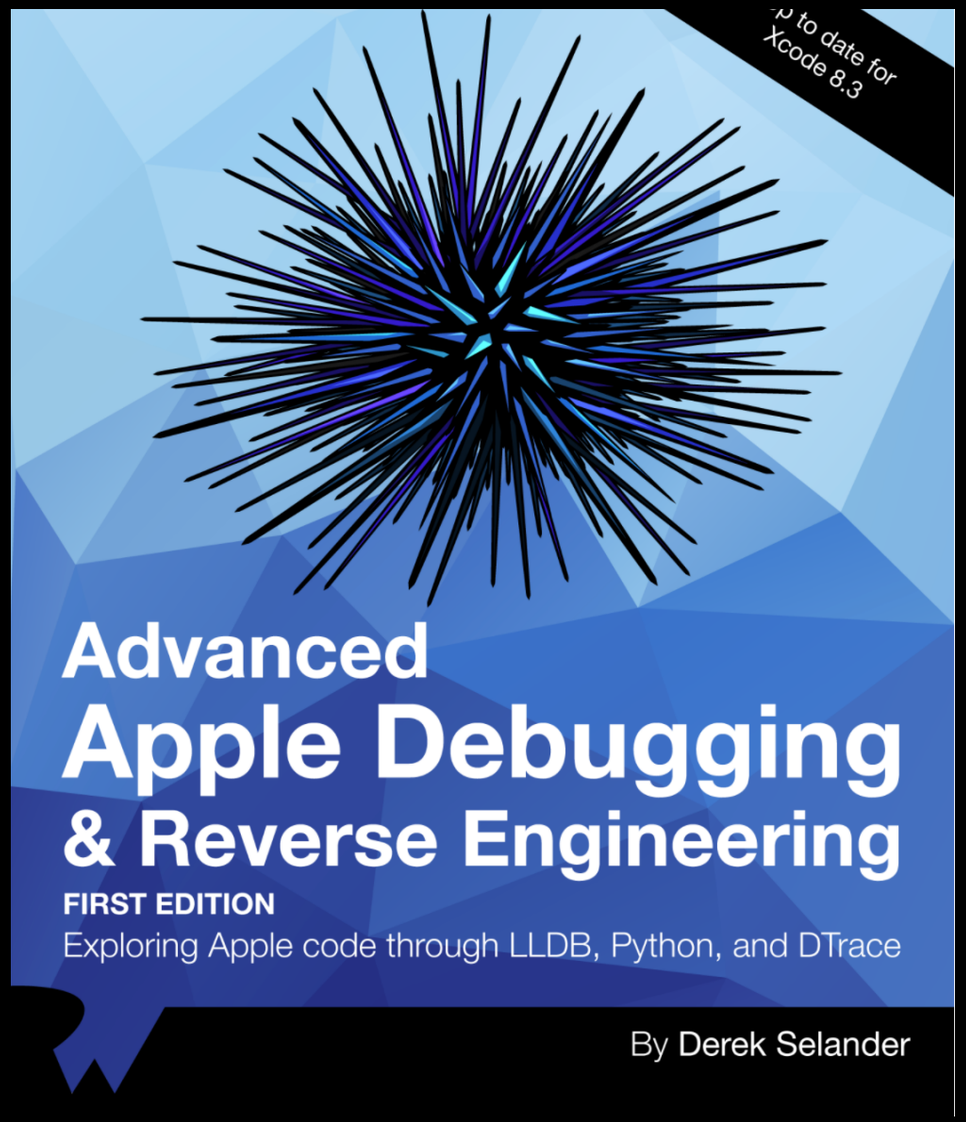


The Tools

- https://github.com/DerekSelander/lldb

- **search** - enumerates the heap to find objects of a certain type of class

- **lookup** - search for functions, *globals variables, strings



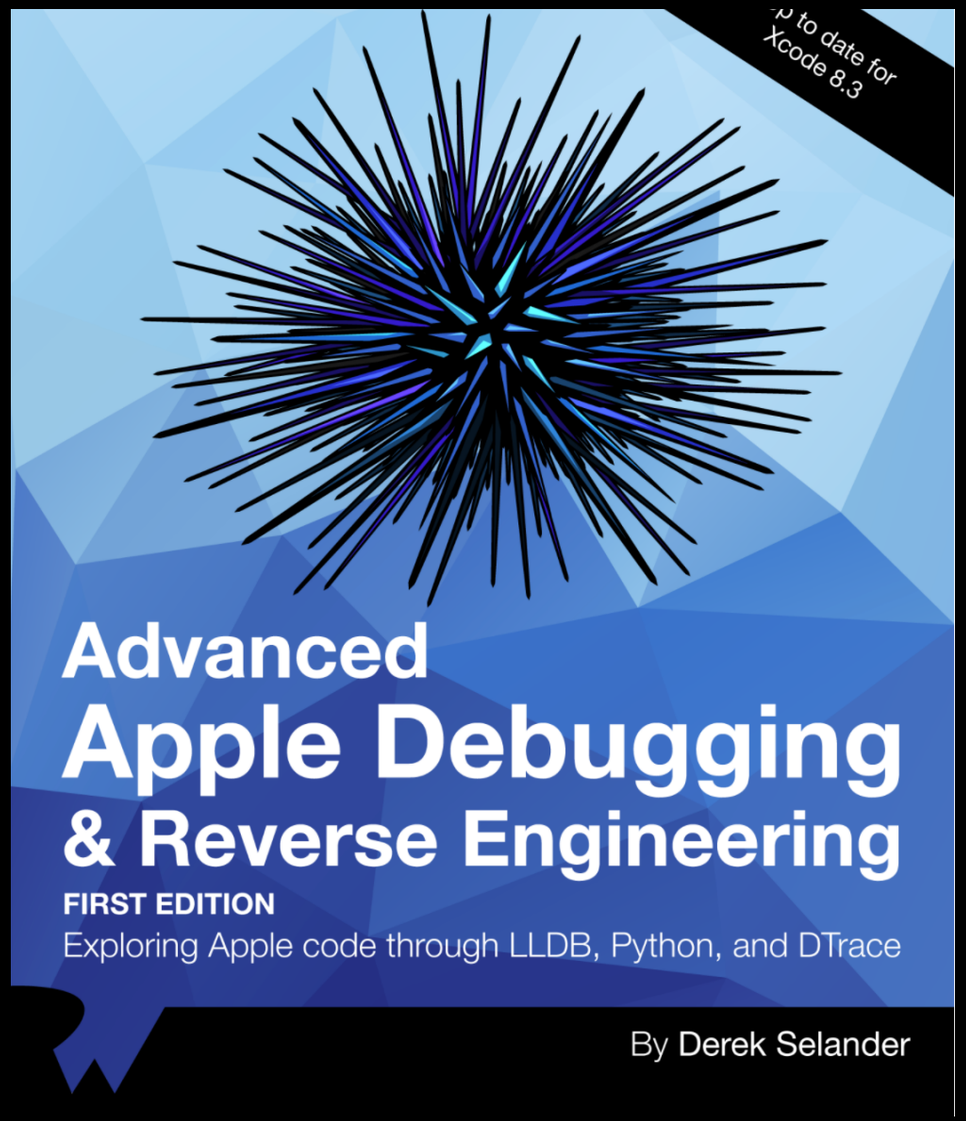
The Tools

- https://github.com/DerekSelander/lldb

- **search** - enumerates the heap to find objects of a certain type of class

- **lookup** - search for functions, *globals variables, strings

- **enable_logging**, **msl** - Enables MallocStackLogging while proc is running, find where an object was created



x86_64 ASM

x86_64 ASM

- **RIP**, the instruction pointer points to where code is executing

```
0x10c7debea <+0>:    push   rbp
0x10c7debeb <+1>:    mov    rbp,  rsp
0x10c7debee <+4>:    push   r15
0x10c7debf0 <+6>:    push   r14
0x10c7debf2 <+8>:    push   r13
0x10c7debf4 <+10>:   push   r12
0x10c7debf6 <+12>:   push   rbx
0x10c7debf7 <+13>:   sub    rsp,  0x98
0x10c7debfe <+20>:   mov    r12,  rdi
0x10c7dec01 <+23>:   xor    eax,  eax
0x10c7dec03 <+25>:   call   0x10c6d06f2  ; _
0x10c7dec08 <+30>:   mov    rdi,  rax
0x10c7dec0b <+33>:   call   0x10c836b8e  ; (
```

x86_64 ASM

- **RIP**, the instruction pointer points to where code is executing
- **RSP** points to the head of the stack

```
0x10c7debea <+0>:    push   rbp
0x10c7debeb <+1>:    mov    rbp,  rsp
0x10c7debee <+4>:    push   r15
0x10c7debf0 <+6>:    push   r14
0x10c7debf2 <+8>:    push   r13
0x10c7debf4 <+10>:   push   r12
0x10c7debf6 <+12>:   push   rbx
0x10c7debf7 <+13>:   sub    rsp,  0x98
0x10c7debfe <+20>:   mov    r12,  rdi
0x10c7dec01 <+23>:   xor    eax,  eax
0x10c7dec03 <+25>:   call   0x10c6d06f2  ; _
0x10c7dec08 <+30>:   mov    rdi,  rax
0x10c7dec0b <+33>:   call   0x10c836b8e  ; (
```

x86_64 ASM

- **RIP**, the instruction pointer points to where code is executing
- **RSP** points to the head of the stack
- **RBP** is a helper to **RSP**

```
0x10c7debea <+0>:    push   rbp
0x10c7debeb <+1>:    mov    rbp,  rsp
0x10c7debee <+4>:    push   r15
0x10c7debf0 <+6>:    push   r14
0x10c7debf2 <+8>:    push   r13
0x10c7debf4 <+10>:   push   r12
0x10c7debf6 <+12>:   push   rbx
0x10c7debf7 <+13>:   sub    rsp,  0x98
0x10c7debfe <+20>:   mov    r12,  rdi
0x10c7dec01 <+23>:   xor    eax,  eax
0x10c7dec03 <+25>:   call   0x10c6d06f2  ; _
0x10c7dec08 <+30>:   mov    rdi,  rax
0x10c7dec0b <+33>:   call   0x10c836b8e  ; (
```

x86_64 ASM

- **RIP**, the instruction pointer points to where code is executing
- **RSP** points to the head of the stack
- **RBP** is a helper to **RSP**
- Register calling convention: **RDI**, **RSI**, **RDX**, **RCX**, **R8**, **R9** then stack

```
0x10c7debea <+0>:    push   rbp
0x10c7debeb <+1>:    mov    rbp,  rsp
0x10c7debee <+4>:    push   r15
0x10c7debf0 <+6>:    push   r14
0x10c7debf2 <+8>:    push   r13
0x10c7debf4 <+10>:   push   r12
0x10c7debf6 <+12>:   push   rbx
0x10c7debf7 <+13>:   sub    rsp,  0x98
0x10c7debfe <+20>:   mov    r12,  rdi
0x10c7dec01 <+23>:   xor    eax,  eax
0x10c7dec03 <+25>:   call   0x10c6d06f2  ; _
0x10c7dec08 <+30>:   mov    rdi,  rax
0x10c7dec0b <+33>:   call   0x10c836b8e  ; (
```

x86_64 ASM

- **RIP**, the instruction pointer points to where code is executing
- **RSP** points to the head of the stack
- **RBP** is a helper to **RSP**
- Register calling convention: **RDI**, **RSI**, **RDX**, **RCX**, **R8**, **R9** then stack
- Return value will be in **RAX**

```
0x10c7debea <+0>:    push   rbp
0x10c7debeb <+1>:    mov    rbp,  rsp
0x10c7debee <+4>:    push   r15
0x10c7debf0 <+6>:    push   r14
0x10c7debf2 <+8>:    push   r13
0x10c7debf4 <+10>:   push   r12
0x10c7debf6 <+12>:   push   rbx
0x10c7debf7 <+13>:   sub    rsp,  0x98
0x10c7debfe <+20>:   mov    r12,  rdi
0x10c7dec01 <+23>:   xor    eax,  eax
0x10c7dec03 <+25>:   call   0x10c6d06f2  ; _
0x10c7dec08 <+30>:   mov    rdi,  rax
0x10c7dec0b <+33>:   call   0x10c836b8e  ; (
```

x86_64 ASM

- **RIP**, the instruction pointer points to where code is executing
- **RSP** points to the head of the stack
- **RBP** is a helper to **RSP**
- Register calling convention: **RDI**, **RSI**, **RDX**, **RCX**, **R8**, **R9** then stack
- Return value will be in **RAX**
- **R** stands for 64-bits, sometimes not all 64 bits are needed (**E == 32**)

```
0x10c7debea <+0>:    push   rbp
0x10c7debeb <+1>:    mov    rbp,  rsp
0x10c7debee <+4>:    push   r15
0x10c7debf0 <+6>:    push   r14
0x10c7debf2 <+8>:    push   r13
0x10c7debf4 <+10>:   push   r12
0x10c7debf6 <+12>:   push   rbx
0x10c7debf7 <+13>:   sub    rsp,  0x98
0x10c7debfe <+20>:   mov    r12,  rdi
0x10c7dec01 <+23>:   xor    eax,  eax
0x10c7dec03 <+25>:   call   0x10c6d06f2  ; _
0x10c7dec08 <+30>:   mov    rdi,  rax
0x10c7dec0b <+33>:   call   0x10c836b8e  ; (
```

x86_64 ASM

- **RIP**, the instruction pointer points to where code is executing
- **RSP** points to the head of the stack
- **RBP** is a helper to **RSP**
- Register calling convention: **RDI**, **RSI**, **RDX**, **RCX**, **R8**, **R9** then stack
- Return value will be in **RAX**
- **R** stands for 64-bits, sometimes not all 64 bits are needed (**E == 32**)
- A **call** opcode means the **RIP** is executing a function

```
0x10c7debea <+0>:    push   rbp
0x10c7debeb <+1>:    mov    rbp,  rsp
0x10c7debee <+4>:    push   r15
0x10c7debf0 <+6>:    push   r14
0x10c7debf2 <+8>:    push   r13
0x10c7debf4 <+10>:   push   r12
0x10c7debf6 <+12>:   push   rbx
0x10c7debf7 <+13>:   sub    rsp,  0x98
0x10c7debfe <+20>:   mov    r12,  rdi
0x10c7dec01 <+23>:   xor    eax,  eax
0x10c7dec03 <+25>:   call   0x10c6d06f2  ; _
0x10c7dec08 <+30>:   mov    rdi,  rax
0x10c7dec0b <+33>:   call   0x10c836b8e  ; (
```

Objective-C & ASM

Objective-C & ASM

- ```
  UIViewController *vc = [UIViewController new];
  UIView *v = [vc view];
  ```
  ```

Objective-C & ASM

- `UIViewController *vc = [UIViewController new];`
  `UIView *v = [vc view];`

- `vc_ref = obj_msgSend(UIViewController_CLASSREF, “new”)`
  `v_ref = obj_msgSend(vc_ref, “view”)`

Objective-C & ASM

- `UIViewController *vc = [UIViewController new];`
  `UIView *v = [vc view];`

- `vc_ref = obj_msgSend(UIViewController_CLASSREF, “new”)`
  `v_ref = obj_msgSend(vc_ref, “view”)`

- `RDI = UIViewController_CLASSREF`
  `RSI = “new”`
  `RAX = obj_msgSend(RDI, RSI)`
  `RDI = RAX`
  `RSI = “view”`
  `RAX = obj_msgSend(RDI, RSI)`

Objective-C & ASM

UIViewController *vc = [UIViewController new];
UIView *v = [vc view];

```
mov    rdi,  qword ptr [rip + 0x3f0a]  ; (__DATA.__objc_classrefs) UIViewController
mov    rsi,  qword ptr [rip + 0x3eb3]  ; "new"
mov    r14,  qword ptr [rip + 0x2e44] (void *)0x0000000102bf1940: objc_msgSend ; 0x102

call   r14
mov    rbx,  rax
mov    rsi,  qword ptr [rip + 0x3ea7]  ; "view"
mov    rdi,  rbx
call   r14
```

C, Swift & ASM

*If the Swift/Objective-C interoperability is not needed (i.e. class/functions not marked as @dynamic)

**Object Oriented languages will typically reserve the first register param as the instance or class parameter in a function

C, Swift & ASM

- Objective-C's objc_msgSend not required, can be a direct **call***

*If the Swift/Objective-C interoperability is not needed (i.e. class/functions not marked as @dynamic)

**Object Oriented languages will typically reserve the first register param as the instance or class parameter in a function

C, Swift & ASM

- Objective-C's objc_msgSend not required, can be a direct **call***
- **RDI** & **RSI** can be used for other parameters**

*If the Swift/Objective-C interoperability is not needed (i.e. class/functions not marked as @dynamic)

**Object Oriented languages will typically reserve the first register param as the instance or class parameter in a function

C, Swift & ASM

- Objective-C's objc_msgSend not required, can be a direct **call***
- **RDI** & **RSI** can be used for other parameters**
- Swift often finds the class (which contains metadata on functions specific to the class) and **call**s from there

```
class ASwiftClass {
    class func aClassFunc() { print("woot") } ; ASwiftClass.aClassFunc()
}
```

*If the Swift/Objective-C interoperability is not needed (i.e. class/functions not marked as @dynamic)

**Object Oriented languages will typically reserve the first register param as the instance or class parameter in a function

C, Swift & ASM

- Objective-C's objc_msgSend not required, can be a direct **call***
- **RDI** & **RSI** can be used for other parameters**
- Swift often finds the class (which contains metadata on functions specific to the class) and **call**s from there

```
class ASwiftClass {
    class func aClassFunc() { print("woot") } ; ASwiftClass.aClassFunc()
}
```

```
call  0x10da96820  ; type metadata accessor for TEST.ASwiftClass
mov   rsi,  qword ptr [rax + 0x50]
mov   r13,  rax
call  rsi
```

*If the Swift/Objective-C interoperability is not needed (i.e. class/functions not marked as @dynamic)

**Object Oriented languages will typically reserve the first register param as the instance or class parameter in a function

mach_header_64

```
#include <mach-o/loader.h>
```

```
/*
 * The 64-bit mach header appears at the very beginning of object files for
 * 64-bit architectures.
 */
struct mach_header_64 {
    uint32_t    magic;      /* mach magic number identifier */
    cpu_type_t  cputype;    /* cpu specifier */
    cpu_subtype_t   cpusubtype; /* machine specifier */
    uint32_t    filetype;   /* type of file */
    uint32_t    ncmds;      /* number of load commands */
    uint32_t    sizeofcmds; /* the size of all the load commands */
    uint32_t    flags;      /* flags */
    uint32_t    reserved;   /* reserved */
};

/* Constant for the magic field of the mach_header_64 (64-bit architectures) */
#define MH_MAGIC_64 0xfeedfacf /* the 64-bit mach magic number */
#define MH_CIGAM_64 0xcffaedfe /* NXSwapInt(MH_MAGIC_64) */
```

mach_header_64

mach_header_64

```
:~ xxd  /bin/ls  | head -5
00000000: cffa edfe 0700 0001 0300 0080 0200 0000  ................
00000010: 1200 0000 0807 0000 8500 2000 0000 0000  .......... .....
00000020: 1900 0000 4800 0000 5f5f 5041 4745 5a45  ....H..__PAGEZE
00000030: 524f 0000 0000 0000 0000 0000 0000 0000  RO..............
00000040: 0000 0000 0100 0000 0000 0000 0000 0000  ................
```

mach_header_64

```
:~ xxd  /bin/ls  | head -5
00000000: cffa edfe 0700 0001 0300 0080 0200 0000  ................
00000010: 1200 0000 0807 0000 8500 2000 0000 0000  .......... .....
00000020: 1900 0000 4800 0000 5f5f 5041 4745 5a45  ....H..._PAGEZE
00000030: 524f 0000 0000 0000 0000 0000 0000 0000  RO..............
00000040: 0000 0000 0100 0000 0000 0000 0000 0000  ................
```

```
:~ xxd -e /bin/ls  | head -5
00000000: feedfacf 01000007 80000003 00000002  ................
00000010: 00000012 00000708 00200085 00000000  .......... .....
00000020: 00000019 00000048 41505f5f 455a4547  ....H..._PAGEZE
00000030: 00004f52 00000000 00000000 00000000  RO..............
00000040: 00000000 00000001 00000000 00000000  ................
```

mach_header_64

```
struct mach_header_64 {
    uint32_t    magic;
    cpu_type_t  cputype;
    cpu_subtype_t   cpusubtype;
    uint32_t    filetype;
    uint32_t    ncmds;
    uint32_t    sizeofcmds;
    uint32_t    flags;
    uint32_t    reserved;
};
```

```
▶ :~ xxd -e /bin/ls  | head -5
00000000: feedfacf 01000007 80000003 00000002  ................
00000010: 00000012 00000708 00200085 00000000  ......... .....
00000020: 00000019 00000048 41505f5f 455a4547  ....H...__PAGEZE
00000030: 00004f52 00000000 00000000 00000000  RO..............
00000040: 00000000 00000001 00000000 00000000  ................
```

mach_header_64

```
struct mach_header_64 {
    uint32_t    magic;
    cpu_type_t  cputype;
    cpu_subtype_t   cpusubtype;
    uint32_t    filetype;
    uint32_t    ncmds;
    uint32_t    sizeofcmds;
    uint32_t    flags;
    uint32_t    reserved;
};
```

```
:~ xxd -e /bin/ls  | head -5
00000000: feedfacf 01000007 80000003 00000002  ................
00000010: 00000012 00000708 00200085 00000000  ......... .....
00000020: 00000019 00000048 41505f5f 455a4547  ....H...__PAGEZE
00000030: 00004f52 00000000 00000000 00000000  RO..............
00000040: 00000000 00000001 00000000 00000000  ................
```

```
#define MH_EXECUTE  0x2     /* demand paged executable file */
```

mach_header_64

```
struct mach_header_64 {
    uint32_t    magic;
    cpu_type_t  cputype;
    cpu_subtype_t   cpusubtype;
    uint32_t    filetype;
    uint32_t    ncmds;
    uint32_t    sizeofcmds;
    uint32_t    flags;
    uint32_t    reserved;
};
```

```
▶ :~ xxd -e /bin/ls  | head -5
00000000: feedfacf 01000007 80000003 00000002  ................
00000010: 00000012 00000708 00200085 00000000  ......... .....
00000020: 00000019 00000048 41505f5f 455a4547  ....H...__PAGEZE
00000030: 00004f52 00000000 00000000 00000000  RO..............
00000040: 00000000 00000001 00000000 00000000  ................
```

```
▶ :~ otool -l /bin/ls | grep magic -A1
      magic cputype cpusubtype  caps    filetype ncmds sizeofcmds      flags
0xfeedfacf 16777223          3  0x80           2    18       1800 0x00200085
```

Segments & Sections

Segments & Sections

- Every compiled executable on your Apple *OS device (even kernel!!!) has this format

Segments & Sections

- Every compiled executable on your Apple *OS device (even kernel!!!) has this format

- Executable is divided up into different **Segments**

```
[0x0000000000000-0x000001000000] 0x0100000000 ls`__PAGEZERO
[0x000001000000-0x000001000005000] 0x0000005000 ls`__TEXT
[0x000001000005000-0x000001000006000] 0x0000001000 ls`__DATA
[0x000001000006000-0x00000010000a000] 0x0000004000 ls`__LINKEDIT
```

Segments & Sections

- Every compiled executable on your Apple *OS device (even kernel!!!) has this format
- Executable is divided up into different **Segments**

```
[0x0000000000000-0x000001000000] 0x0100000000 ls`__PAGEZERO
[0x000001000000-0x000001000005000] 0x0000005000 ls`__TEXT
[0x000001000005000-0x000001000006000] 0x0000001000 ls`__DATA
[0x000001000006000-0x00000010000a000] 0x0000004000 ls`__LINKEDIT
```

- These Segments can have 0 or more **Sections**

```
[0x00000100000f00-0x000001000441f] 0x000000351f ls`__TEXT.__text
[0x000001000004420-0x00001000045e8] 0x00000001c8 ls`__TEXT.__stubs
[0x00001000045e8-0x00001000048f0] 0x0000000308 ls`__TEXT.__stub_helper
[0x00001000048f0-0x00000100004ae8] 0x00000001f8 ls`__TEXT.__const
[0x00000100004ae8-0x00000100004f66] 0x000000047e ls`__TEXT.__cstring
[0x00000100004f68-0x00000100004ffc] 0x0000000094 ls`__TEXT.__unwind_info
```

Segments & Sections

- Every compiled executable on your Apple *OS device (even kernel!!!) has this format

- Executable is divided up into different **Segments**

```
[0x0000000000000-0x000001000000] 0x0100000000 ls`__PAGEZERO
[0x000001000000-0x000001000005000] 0x0000005000 ls`__TEXT
[0x000001000005000-0x000001000006000] 0x0000001000 ls`__DATA
[0x000001000006000-0x00000010000a000] 0x0000004000 ls`__LINKEDIT
```

- These Segments can have 0 or more **Sections**

```
[0x00000100000f00-0x000001000441f] 0x000000351f ls`__TEXT.__text
[0x000001000004420-0x00001000045e8] 0x00000001c8 ls`__TEXT.__stubs
[0x00001000045e8-0x00001000048f0] 0x0000000308 ls`__TEXT.__stub_helper
[0x00001000048f0-0x000001000004ae8] 0x00000001f8 ls`__TEXT.__const
[0x000001000004ae8-0x000001000004f66] 0x000000047e ls`__TEXT.__cstring
[0x000001000004f68-0x000001000004ffc] 0x0000000094 ls`__TEXT.__unwind_info
```

- Understanding these Sections and how they work inside the proc can give you A LOT of power.

Amusing Sections

```
[0x0000010694ae90-0x00000106c38b59] 0x00002edcc9 GoDaddyVoice`__TEXT.__text
[0x00000106c38b5a-0x00000106c3a1e0] 0x0000001686 GoDaddyVoice`__TEXT.__stubs
[0x00000106c3a1e0-0x00000106c3c77a] 0x000000259a GoDaddyVoice`__TEXT.__stub_helper
[0x00000106c3c780-0x00000106c48058] 0x000000b8d8 GoDaddyVoice`__TEXT.__const
[0x00000106c48060-0x00000106c66c12] 0x000001ebb2 GoDaddyVoice`__TEXT.__cstring
[0x00000106c66c12-0x00000106c79741] 0x0000012b2f GoDaddyVoice`__TEXT.__objc_methname
[0x00000106c79741-0x00000106c7a135] 0x00000009f4 GoDaddyVoice`__TEXT.__objc_classname
[0x00000106c7a135-0x00000106c7bec3] 0x0000001d8e GoDaddyVoice`__TEXT.__objc_methtype
[0x00000106c7bed0-0x00000106c88283] 0x000000c3b3 GoDaddyVoice`__TEXT.__swift3_typeref
[0x00000106c88290-0x00000106c8c0b2] 0x0000003e22 GoDaddyVoice`__TEXT.__swift3_reflstr
[0x00000106c8c0b4-0x00000106c8ff84] 0x0000003ed0 GoDaddyVoice`__TEXT.__swift3_fieldmd
[0x00000106c8ff88-0x00000106c90268] 0x00000002e0 GoDaddyVoice`__TEXT.__swift2_types
[0x00000106c90268-0x00000106c9208c] 0x0000001e24 GoDaddyVoice`__TEXT.__swift3_capture
[0x00000106c92090-0x00000106c93c90] 0x0000001c00 GoDaddyVoice`__TEXT.__swift2_proto
[0x00000106c93c90-0x00000106c94248] 0x00000005b8 GoDaddyVoice`__TEXT.__swift3_assocty
[0x00000106c94248-0x00000106c9439c] 0x0000000154 GoDaddyVoice`__TEXT.__swift3_builtin
[0x00000106c943a0-0x00000106c9517a] 0x0000000dda GoDaddyVoice`__TEXT.__ustring
[0x00000106c9517c-0x00000106c9601c] 0x0000000ea0 GoDaddyVoice`__TEXT.__gcc_except_tab
[0x00000106c9601c-0x00000106c962c8] 0x00000002ac GoDaddyVoice`__TEXT.__entitlements
[0x00000106c962c8-0x00000106c9d130] 0x0000006e68 GoDaddyVoice`__TEXT.__unwind_info
[0x00000106c9d130-0x00000106c9e000] 0x0000000ed0 GoDaddyVoice`__TEXT.__eh_frame
```

Amusing Sections

```
[0x00000106c9e000-0x00000106c9e010] 0x0000000010 GoDaddyVoice`__DATA.__nl_symbol_ptr
[0x00000106c9e010-0x00000106c9e8d0] 0x00000008c0 GoDaddyVoice`__DATA.__got
[0x00000106c9e8d0-0x00000106ca06d8] 0x0000001e08 GoDaddyVoice`__DATA.__la_symbol_ptr
[0x00000106ca06d8-0x00000106ca06e0] 0x0000000008 GoDaddyVoice`__DATA.__mod_init_func
[0x00000106ca06e0-0x00000106caf820] 0x000000f140 GoDaddyVoice`__DATA.__const
[0x00000106caf820-0x00000106cb7a20] 0x0000008200 GoDaddyVoice`__DATA.__cfstring
[0x00000106cb7a20-0x00000106cb8110] 0x00000006f0 GoDaddyVoice`__DATA.__objc_classlist
[0x00000106cb8110-0x00000106cb8330] 0x0000000220 GoDaddyVoice`__DATA.__objc_catlist
[0x00000106cb8330-0x00000106cb84b0] 0x0000000180 GoDaddyVoice`__DATA.__objc_protolist
[0x00000106cb84b0-0x00000106cb84b8] 0x0000000008 GoDaddyVoice`__DATA.__objc_imageinfo
[0x00000106cb84b8-0x00000106cf2e18] 0x000003a960 GoDaddyVoice`__DATA.__objc_const
[0x00000106cf2e18-0x00000106cf6b50] 0x0000003d38 GoDaddyVoice`__DATA.__objc_selrefs
[0x00000106cf6b50-0x00000106cf6c30] 0x00000000e0 GoDaddyVoice`__DATA.__objc_protorefs
[0x00000106cf6c30-0x00000106cf7348] 0x0000000718 GoDaddyVoice`__DATA.__objc_classrefs
[0x00000106cf7348-0x00000106cf75d8] 0x0000000290 GoDaddyVoice`__DATA.__objc_superrefs
[0x00000106cf75d8-0x00000106cf8190] 0x0000000bb8 GoDaddyVoice`__DATA.__objc_ivar
[0x00000106cf8190-0x00000106d00e98] 0x0000008d08 GoDaddyVoice`__DATA.__objc_data
[0x00000106d00e98-0x00000106d07328] 0x0000006490 GoDaddyVoice`__DATA.__llvm_prf_cnts
[0x00000106d07328-0x00000106d191d8] 0x0000011eb0 GoDaddyVoice`__DATA.__llvm_prf_data
[0x00000106d191e0-0x00000106d2306c] 0x0000009e8c GoDaddyVoice`__DATA.__llvm_prf_names
[0x00000106d23070-0x00000106d2aec9] 0x0000007e59 GoDaddyVoice`__DATA.__data
[0x00000106d2aec9-0x00000106d2aec9] 0x0000000000 GoDaddyVoice`__DATA.__llvm_prf_vnds
[0x00000106d2aed0-0x00000106d2df60] 0x0000003090 GoDaddyVoice`__DATA.__bss
[0x00000106d2df60-0x00000106d2e308] 0x00000003a8 GoDaddyVoice`__DATA.__common
```