Mach-O Libre

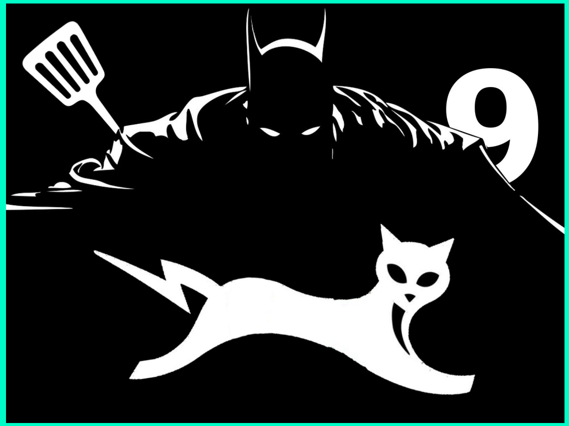
Pile Driving Apple Malware with Static Analysis, Big Data, & Automation

Aaron Stephens & Will Peteroy

Introductions

Aaron

UWT CE/CS
CCDC
Batman’s Kitchen
Neg9



Will

@wepIV
Co-Founder / CEO
ICEBRG.IO

ICEBRG

ICEBRG

www.icebrg.io

Greetz

Thanks everybody!

Elizabeth Walkup
(Stanford)

Andrew Case
(Volatility)

Neil Kandalgaonkar
(Sauce Labs)

Mario De Tore
(ICEBRG)

“Mac Malware Detection via Static File Analysis”

github.com/saucelabs/isign

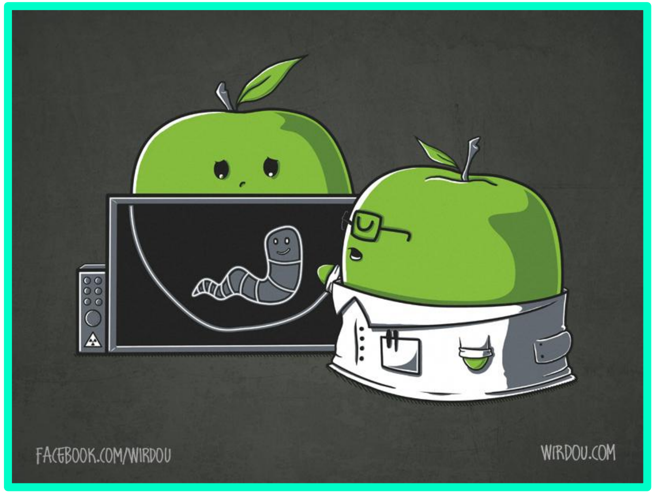
Why are We Here?

1. We (ICEBRG) expand or extend on current tools to handle gaps in our capabilities
2. ICEBRG interns are required to have an “intern project” which challenges them and does something productive for us and for the community
3. Saw the opportunity to build a flexible, performant, open source Mach-O parser for everyone

INTERN

Why Should You Care?

Apple product usage ++
(Even in the Enterprise)
Apple Malware ++
(KeyRaider, YiSpecter, etc.)



How Did it Start?

The format is highly complex and looked like a good rabbit hole

Solved Problem? Sort of... (not really)

There are other parsers.

...some cost money ($$$)

...some require a knowledge of Objective-C / C++

...most have only partial coverage of binary metadata

Areas for improvement

1. Accessibility (python)
2. Coverage / Extensibility
3. Free (Open Source)

Understand the history

Identify key features

Research the format

Work through the code

Research the format

Triumph

Rinse, repeat

Getting the Lay of the Land

TL;DR History Lesson

Thanks Wikipedia!

1977: Berkeley - BSD

1985: CMU - ***Mach*** Kernel

1986: Berkeley - 4.3BSD

1989: NeXT - NeXTSTEP

1993: Berkeley - FreeBSD

1997: Apple acquires NeXT

2000: Apple - Darwin

2001: Apple - OS X 10.0

What is it?

"... a file format for executables, object code, shared libraries, dynamically-loaded code, and coredumps." – Wikipedia

```
$ man Mach-O
The object files produced by the
assembler and link editor are in
Mach-O (Mach object) file format.
```

.... k.

```
The complete description of a
Mach-O file is given in a number
of include files.  The file <mach-
o/loader.h> describes the headers,
<mach-o/nlist.h> describes the
symbol table entries with <mach-
o/stab.h> supplementing it, and
<mach-o/reloc.h> describes the
relocation entries.
```

Where is it Found?

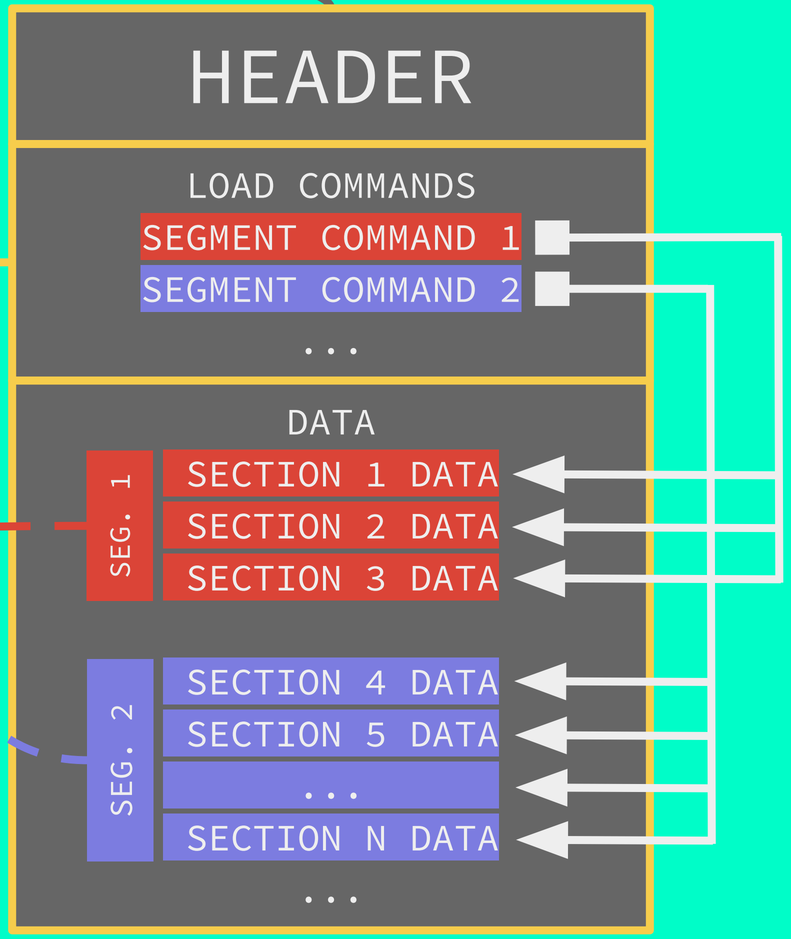
- /Applications/
- /Library/
- /usr/bin/
- /Cores/
- /System/

```
$ file /bin/* | grep 'Mach-O' | wc -l
    39
$ file /sbin/* | grep 'Mach-O' | wc -l
    73
$ file /usr/bin/* | grep 'Mach-O' | wc -l
    913
```

Overall Structure

Higher level binary description: magic, architecture, and flags.

Layout, dependencies, and generic info for the kernel and linker



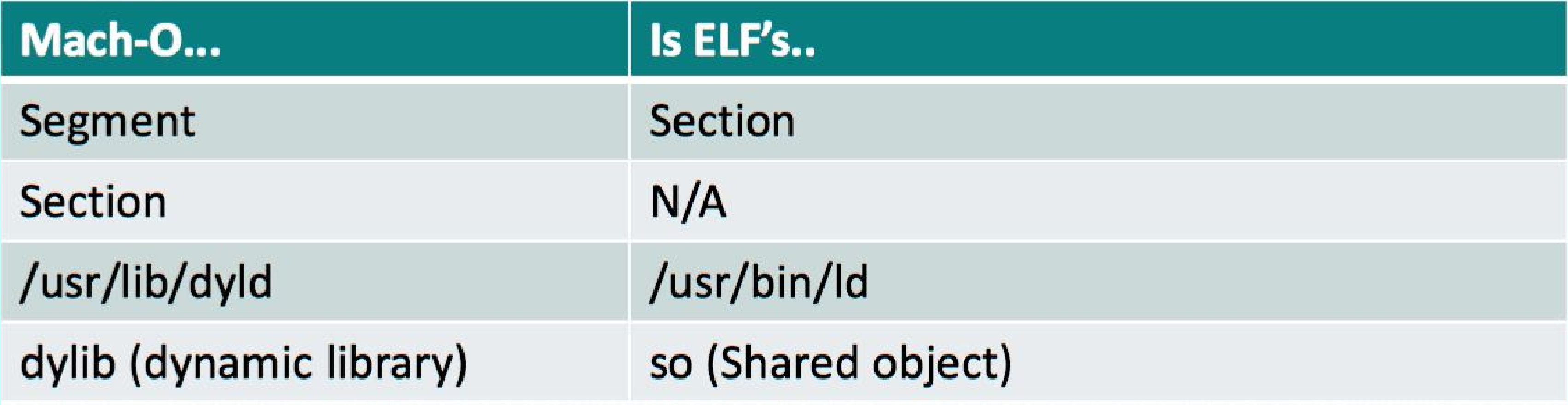
<u>The usual suspects:</u>

- __TEXT
- __DATA
- __OBJC
- __IMPORT
- __LINKEDIT

New segment, but section #’s ***<u>don’t</u>*** reset.

And other fun stuff...

Mach-O File Format VS. Executable and Linkable Format(elf)



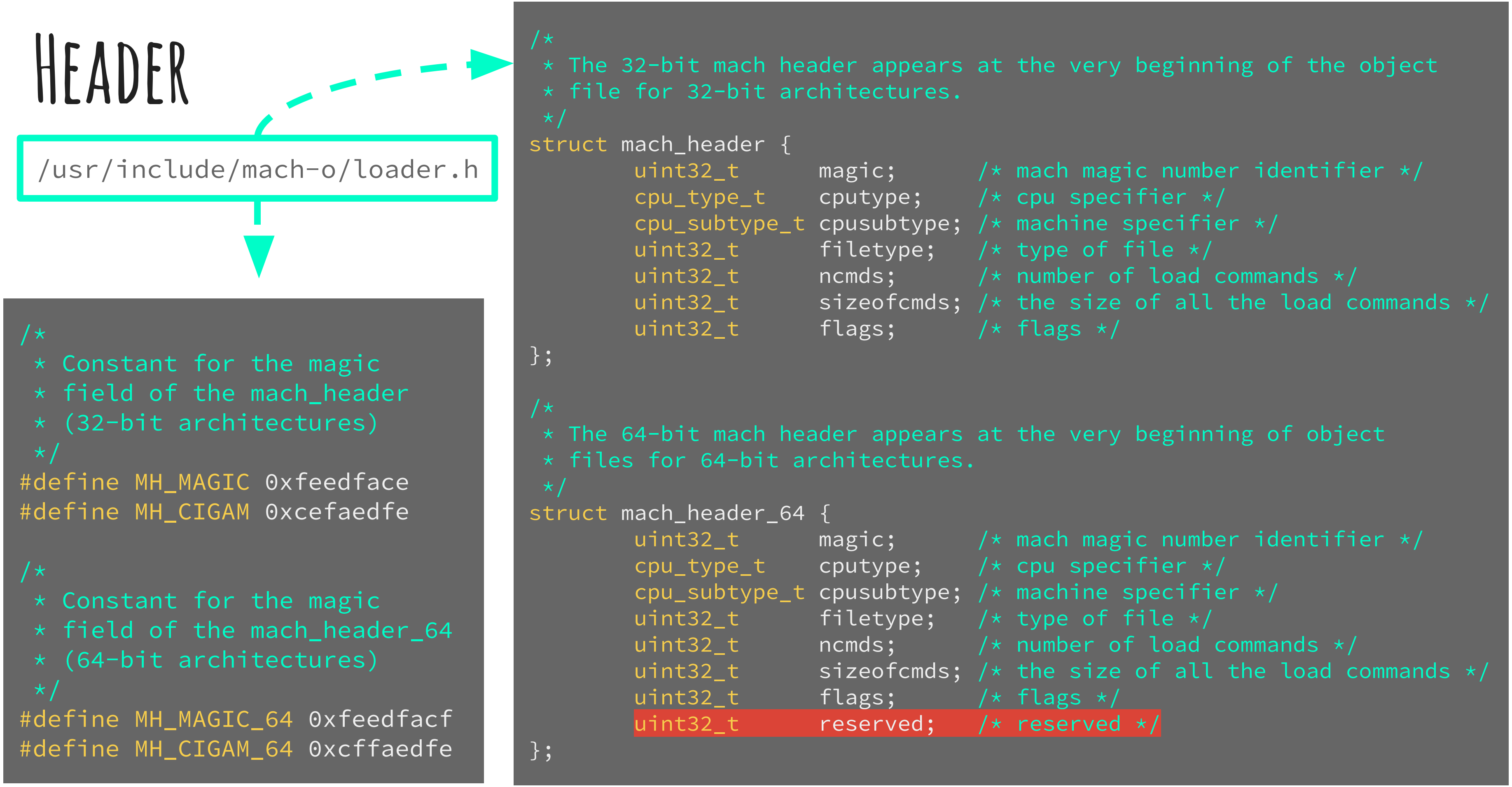
Mach-O...	Is ELF's..
Segment	Section
Section	N/A
/usr/lib/dyld	/usr/bin/ld
dylib (dynamic library)	so (Shared object)

Jonathan Levin - RSA 2015

http://newosxbook.com/articles/CodeSigning.pdf

Down to Details

Header



/usr/include/mach-o/loader.h

```
/*
 * Constant for the magic
 * field of the mach_header
 * (32-bit architectures)
 */
#define MH_MAGIC 0xfeedface
#define MH_CIGAM 0xcefaedfe

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

```
/*
 * The 32-bit mach header appears at the very beginning of the object
 * file for 32-bit architectures.
 */
struct mach_header {
        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 */
};

/*
 * 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 */
};
```

Header: File Types & Flags

What we're focused on

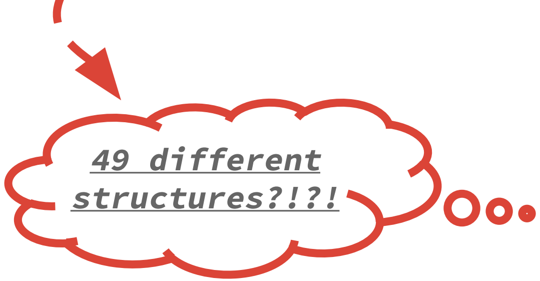
```
/* Constants for the filetype field of the mach_header */
#define MH_OBJECT       0x1  /* relocatable object file */
#define MH_EXECUTE      0x2  /* demand paged executable file */
#define MH_FVMLIB       0x3  /* fixed VM shared library file */
#define MH_CORE         0x4  /* core file */
#define MH_PRELOAD      0x5  /* preloaded executable file */
#define MH_DYLIB        0x6  /* dynamically bound shared library */
#define MH_DYLINKER     0x7  /* dynamic link editor */
#define MH_BUNDLE       0x8  /* dynamically bound bundle file */
#define MH_DYLIB_STUB   0x9  /* shared library stub for static linking only, no section contents */
#define MH_DSYM         0xa  /* companion file with only debug sections */
#define MH_KEXT_BUNDLE  0xb  /* x86_64 kexts */
```

```
/* Constants for the flags field of the mach_header */
#define MH_NOUNDEFS     0x1  /* the object file has no undefined references */
#define MH_INCRLINK     0x2  /* the object file is the output of an incremental link against a base
                                file and can't be link edited again */
#define MH_DYLDLINK     0x4  /* the object file is input for the dynamic linker and can't be
                                statically link edited again */
#define MH_BINDATLOAD   0x8  /* the object file's undefined references are bound by the dynamic
                                linker when loaded. */
#define MH_PREBOUND     0x10 /* the file has its dynamic undefined references prebound. */
...
```

Load Commands

```
#define LC_SEGMENT        0x1  /* segment of this file to be mapped */
#define LC_SEGMENT_64     0x19 /* 64-bit segment of this file to be mapped */
#define LC_SYMTAB         0x2  /* link-edit stab symbol table info */
#define LC_DYSYMTAB       0xb  /* dynamic link-edit symbol table info */
#define LC_LOAD_DYLIB     0xc  /* load a dynamically linked shared library */
#define LC_CODE_SIGNATURE 0x1d /* local of code signature */
...
```

49 different load commands...



49 different structures?!?!

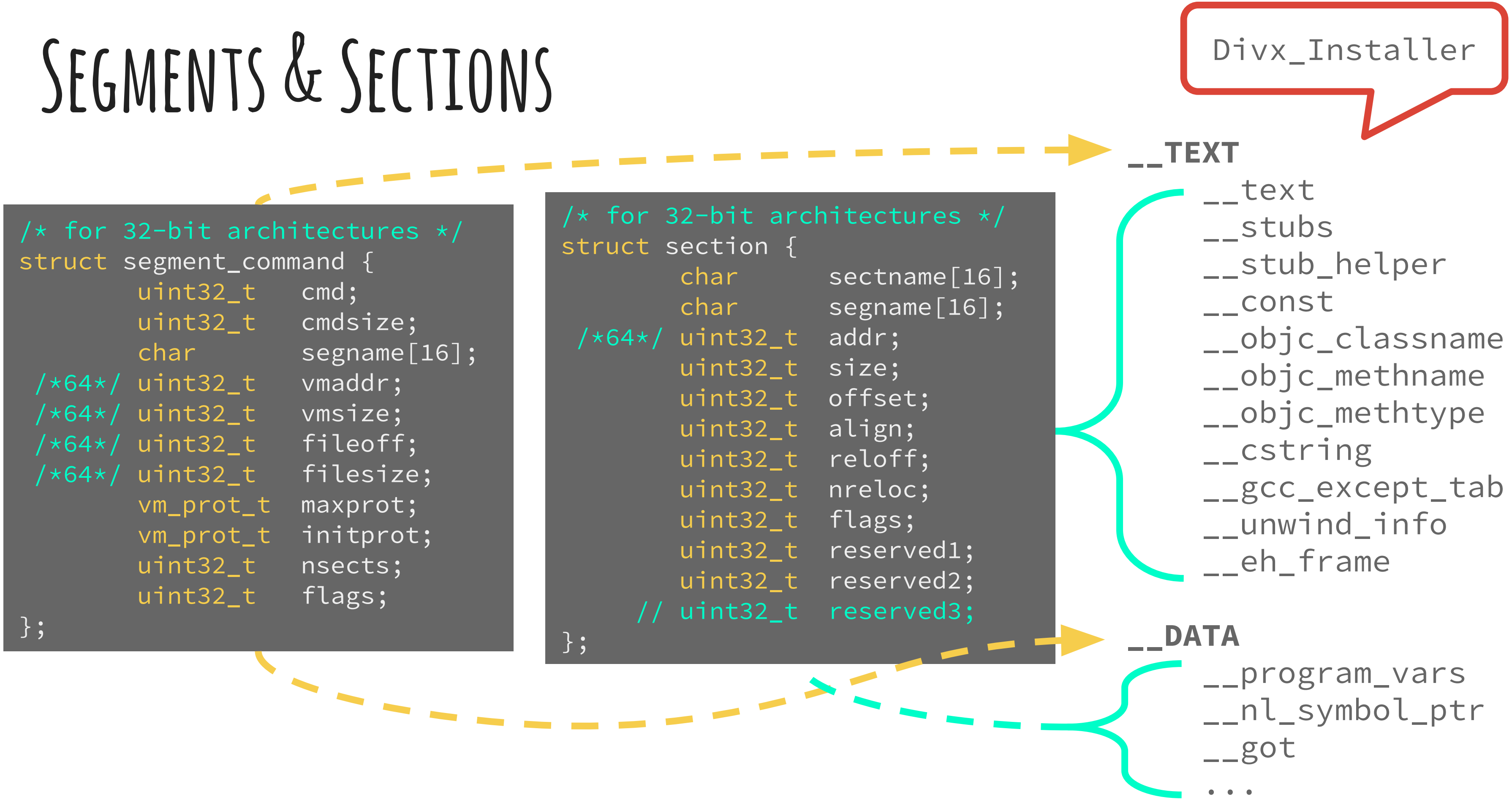
```
/*
 * The load commands directly follow the mach_header.  The total size of all
 * of the commands is given by the sizeofcmds field in the mach_header.  All
 * load commands must have as their first two fields cmd and cmdsize... Each
 * command type has a structure specifically for it.  The cmdsize field is
 * the size in bytes of the particular load command structure plus anything
 * that follows it that is a part of the load command (i.e. section
 * structures, strings, etc.)...  The cmdsize for 32-bit architectures MUST
 * be a multiple of 4 bytes and for 64-bit architectures MUST be a multiple
 * of 8 bytes (these are forever the maximum alignment of any load commands).
 * The padded bytes must be zero.  All tables in the object file must also
 * follow these rules so the file can be memory mapped.  Otherwise the
 * pointers to these tables will not work well or at all on some machines...
 */

struct load_command {
        uint32_t cmd;     /* type of load command */
        uint32_t cmdsize; /* total size of command in bytes */
};
```

... eh, more like 30

linkedit_data_command:

```
  LC_CODE_SIGNATURE
  LC_SEGMENT_SPLIT_INFO
  LC_FUNCTION_STARTS
  LC_DYLIB_CODE_SIGN_DRS
  LC_LINKER_OPTIMIZATION_HINT
```

Segments & Sections



Divx_Installer

```
/* for 32-bit architectures */
struct segment_command {
        uint32_t   cmd;
        uint32_t   cmdsize;
        char       segname[16];
 /*64*/ uint32_t   vmaddr;
 /*64*/ uint32_t   vmsize;
 /*64*/ uint32_t   fileoff;
 /*64*/ uint32_t   filesize;
        vm_prot_t  maxprot;
        vm_prot_t  initprot;
        uint32_t   nsects;
        uint32_t   flags;
};
```

```
/* for 32-bit architectures */
struct section {
        char       sectname[16];
        char       segname[16];
 /*64*/ uint32_t   addr;
        uint32_t   size;
        uint32_t   offset;
        uint32_t   align;
        uint32_t   reloff;
        uint32_t   nreloc;
        uint32_t   flags;
        uint32_t   reserved1;
        uint32_t   reserved2;
     // uint32_t   reserved3;
};
```

__TEXT

- __text
- __stubs
- __stub_helper
- __const
- __objc_classname
- __objc_methname
- __objc_methtype
- __cstring
- __gcc_except_tab
- __unwind_info
- __eh_frame

__DATA

- __program_vars
- __nl_symbol_ptr
- __got
- ...

Symbol Table

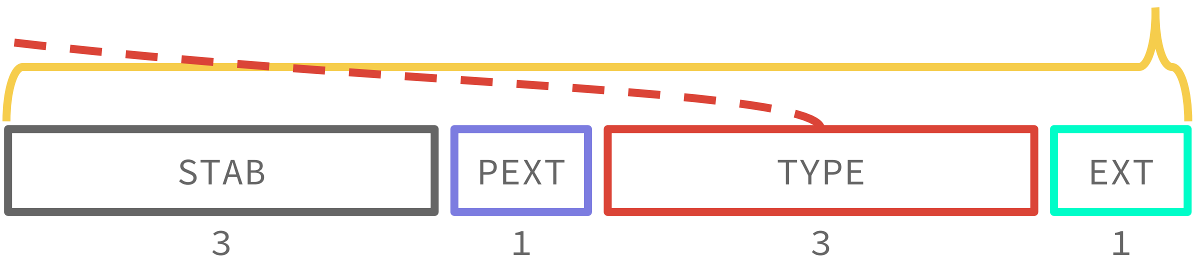
/usr/include/mach-o/nlist.h

```
/*
 * Values for
 * N_TYPE bits of
 * the n_type field.
 */
#define N_UNDF 0x0
#define N_ABS  0x2
#define N_SECT 0xe
#define N_PBUD 0xc
#define N_INDR 0xa
```

```
struct nlist {
        union {
#ifndef __LP64__
                char     *n_name; /* for use when in-core */
#endif
                uint32_t n_strx;  /* index into the string table */
        } n_un;
        uint8_t  n_type;  /* type flag, see below */
        uint8_t  n_sect;  /* section number or NO_SECT */
        int16_t  n_desc;  /* see <mach-o/stab.h> */
 /*64*/ uint32_t n_value; /* value of this symbol (or stab offset) */
};
```

“index” actually means byte offset :P

Indicates “stab” (or debugging) symbol.



Symbols... But what do they mean?!

N_UNDF (0x0): The symbol is undefined. Undefined symbols are symbols referenced in this module but defined in a different module. Set the n_sect field to NO_SECT.

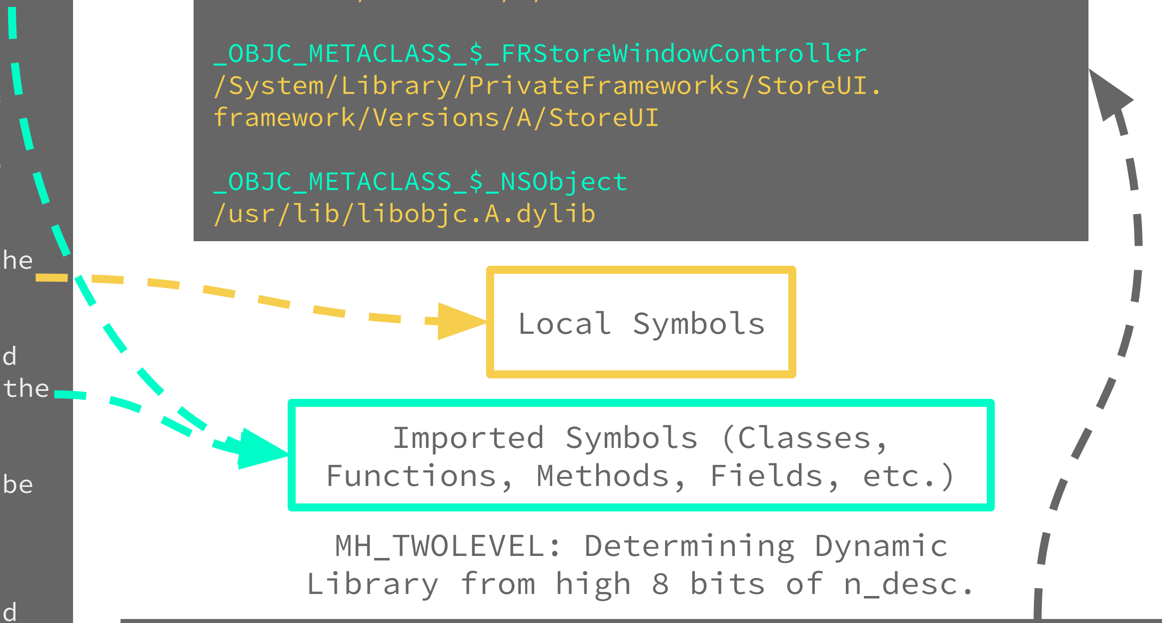
~~N_ABS (0x2)~~: The symbol is absolute. The linker does not update the value of an absolute symbol. Set the n_sect field to NO_SECT.

N_SECT (0xe): The symbol is defined in the section number given in n_sect.

N_PBUD (0xc): The symbol is undefined and the image is using a prebound value for the symbol. Set the n_sect field to NO_SECT.

~~N_INDR (0xa)~~: The symbol is defined to be the same as another symbol. The n_value field is an index into the string table specifying the name of the other symbol. When that symbol is linked, both this and the other symbol point to the same defined type and value.

```
_OBJC_METACLASS_$_FRAppDelegate
/System/Library/PrivateFrameworks/StoreUI.
framework/Versions/A/StoreUI

_OBJC_METACLASS_$_FRStoreWindowController
/System/Library/PrivateFrameworks/StoreUI.
framework/Versions/A/StoreUI

_OBJC_METACLASS_$_NSObject
/usr/lib/libobjc.A.dylib
```


Local Symbols
Imported Symbols (Classes, Functions, Methods, Fields, etc.)

MH_TWOLEVEL: Determining Dynamic Library from high 8 bits of n_desc.

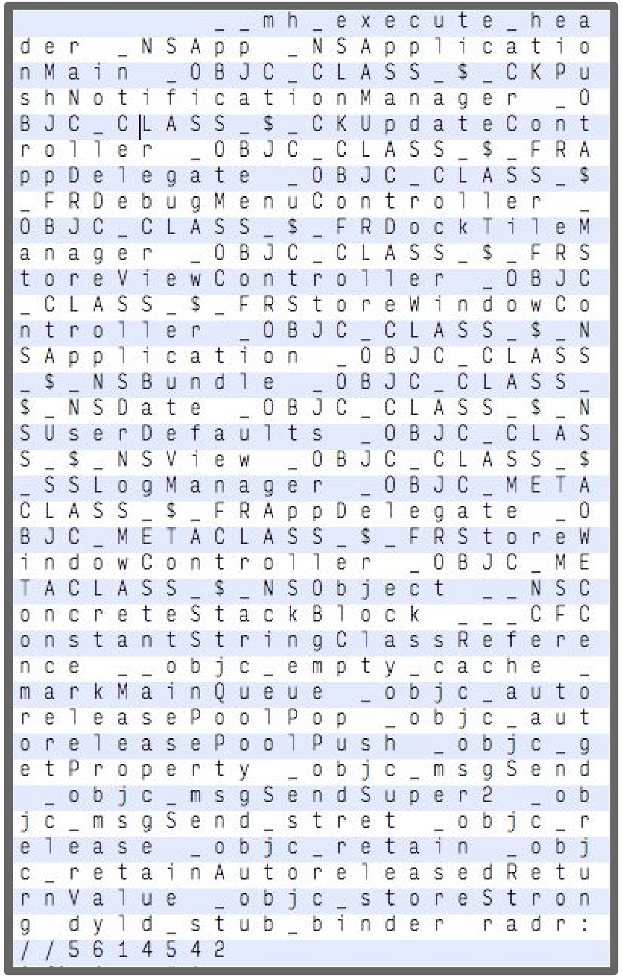
```
#define GET_LIBRARY_ORDINAL(n_desc) (((n_desc) >> 8) & 0xff)
```

http://math-atlas.sourceforge.net/devel/assembly/MachORuntime.pdf

STRING TABLE

```
/*
 * The symtab_command contains the offsets and sizes of
 * the link-edit 4.3BSD "stab" style symbol table
 * information as described in the header files <nlist.h>
 * and <stab.h>.
 */
struct symtab_command {
    uint32_t cmd;      /* LC_SYMTAB */
    uint32_t cmdsize;  /* sizeof(struct symtab_command) */
    uint32_t symoff;   /* symbol table offset */
    uint32_t nsyms;    /* number of symbol table entries */
    uint32_t stroff;   /* string table offset */
    uint32_t strsize;  /* string table size in bytes */
};
```

string table == just a bunch of strings! :D



```
          __mh_execute_hea
der _NSApp _NSApplicatio
nMain _OBJC_CLASS_$_CKPu
shNotificationManager _O
BJC_CLASS_$_CKUpdateCont
roller _OBJC_CLASS_$_FRA
ppDelegate _OBJC_CLASS_$
_FRDebugMenuController _
OBJC_CLASS_$_FRDockTileM
anager _OBJC_CLASS_$_FRS
toreViewController _OBJC
_CLASS_$_FRStoreWindowCo
ntroller _OBJC_CLASS_$_N
SApplication _OBJC_CLASS
_$_NSBundle _OBJC_CLASS_
$_NSDate _OBJC_CLASS_$_N
SUserDefaults _OBJC_CLAS
S_$_NSView _OBJC_CLASS_$
_SSLogManager _OBJC_META
CLASS_$_FRAppDelegate _O
BJC_METACLASS_$_FRStoreW
indowController _OBJC_ME
TACLASS_$_NSObject __NSC
oncreteStackBlock ___CFC
onstantStringClassRefere
nce __objc_empty_cache _
markMainQueue _objc_auto
releasePoolPop _objc_aut
oreleasePoolPush _objc_g
etProperty _objc_msgSend
 _objc_msgSendSuper2 _ob
jc_msgSend_stret _objc_r
elease _objc_retain _obj
c_retainAutoreleasedRetu
rnValue _objc_storeStron
g dyld_stub_binder radr:
//5614542
```

Code Signature

Code Directory

- The "*Bookkeeper*"
- Hashes
  - Executable
  - Info.plist
  - Signature
- Identity

Requirements

- Validation constraints
- Requirement Language (see link below)
- identifier
- certificates

Entitlements

- Permissions
- Capabilities
- iCloud
- Push Notifications
- App Sandboxing

Certificates

- X.509
- CMS SignedData in DER format
- Typically anchored by "Apple Root CA"

https://developer.apple.com/library/mac/documentation/Security/Conceptual/CodeSigningGuide/RequirementLang/RequirementLang.html

Code Signatures: Blobs on Blobs on Blobs…

```
/*
 * Blob types (magic numbers) for blobs used by Code Signing.
 */
enum {
      kSecCodeMagicRequirement =         0xfade0c00, /* single requirement */
      kSecCodeMagicRequirementSet =      0xfade0c01, /* requirement set */
      kSecCodeMagicCodeDirectory =       0xfade0c02, /* CodeDirectory */
      kSecCodeMagicEmbeddedSignature =   0xfade0cc0, /* single-architecture embedded signature */
      kSecCodeMagicDetachedSignature =   0xfade0cc1, /* detached multi-architecture signature */
      kSecCodeMagicEntitlement =         0xfade7171, /* entitlement blob */
      kSecCodeMagicByte =                0xfa        /* shared first byte */
};
```

libsecurity_codesigning/lib/CSCommonPriv.h

lol… wut.

Blob?

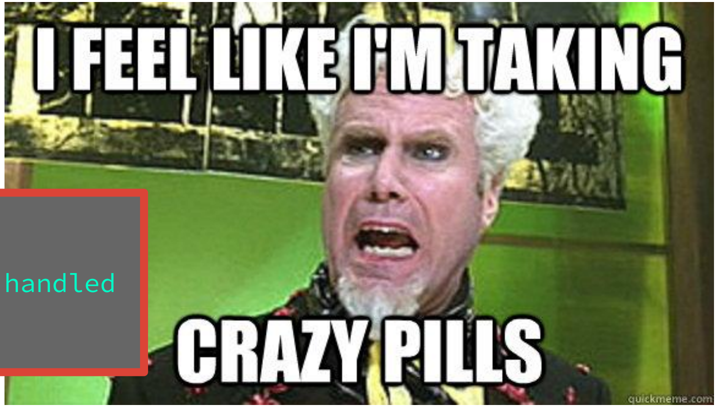
BlobWrapper???

SuperBlob?!?!

opensource.apple.com

libsecurity_utilities/lib/blob.h

```
//
// A generic blob wrapped around arbitrary (flat) binary data.
// This can be used to "regularize" plain binary data, so it can be handled
// as a genuine Blob (e.g. for insertion into a SuperBlob).
//
```

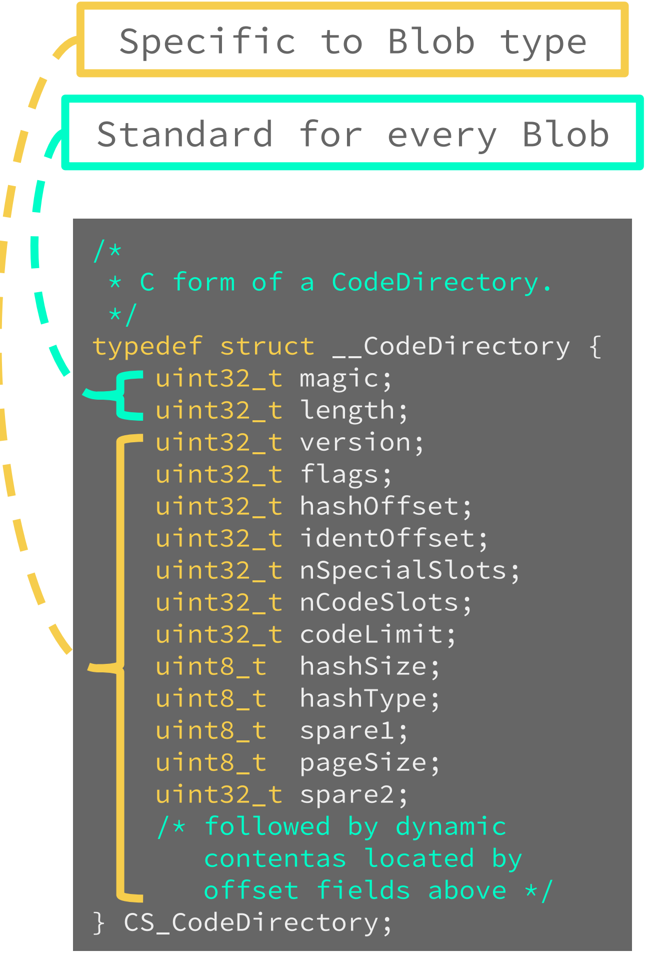


Blobs: They're not so bad...

libsecurity_codesigning/lib/cscdefs.h

```
/*
 * Structure of an embedded-signature SuperBlob
 */
typedef struct __BlobIndex {
      uint32_t type;   /* type of entry */
      uint32_t offset; /* offset of entry */
} CS_BlobIndex;

typedef struct __SuperBlob {
      uint32_t     magic;   /* magic number */
      uint32_t     length;  /* total length of SuperBlob */
      uint32_t     count;   /* number of index entries following */
      CS_BlobIndex index[]; /* (count) entries */
      /* followed by Blobs in no particular order as indicated by
         offsets in index */
} CS_SuperBlob;
```

libsecurity_codesigning/lib/requirements.h
libsecurity_codesigning/lib/sigblob.h

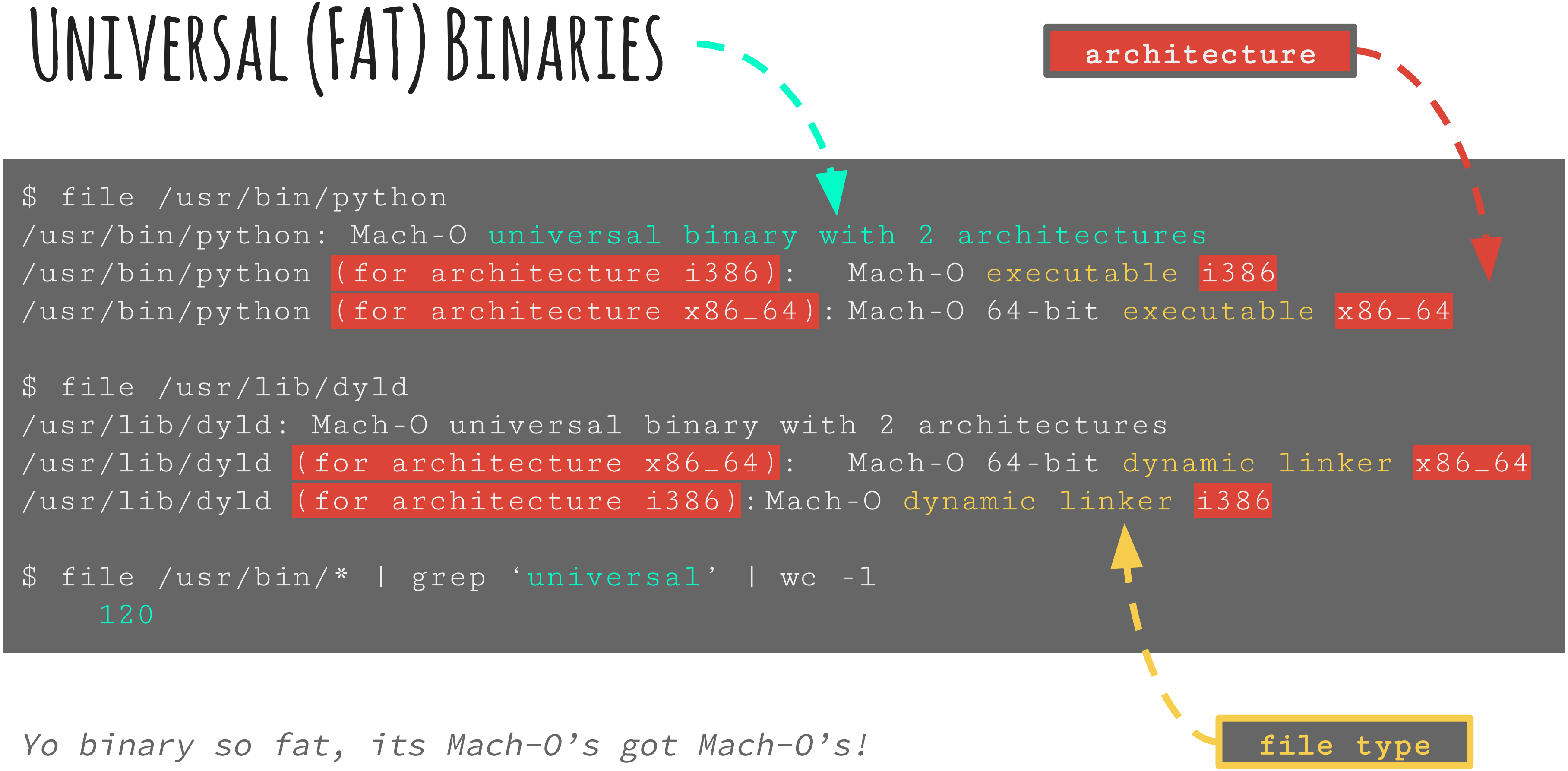


Specific to Blob type

Standard for every Blob

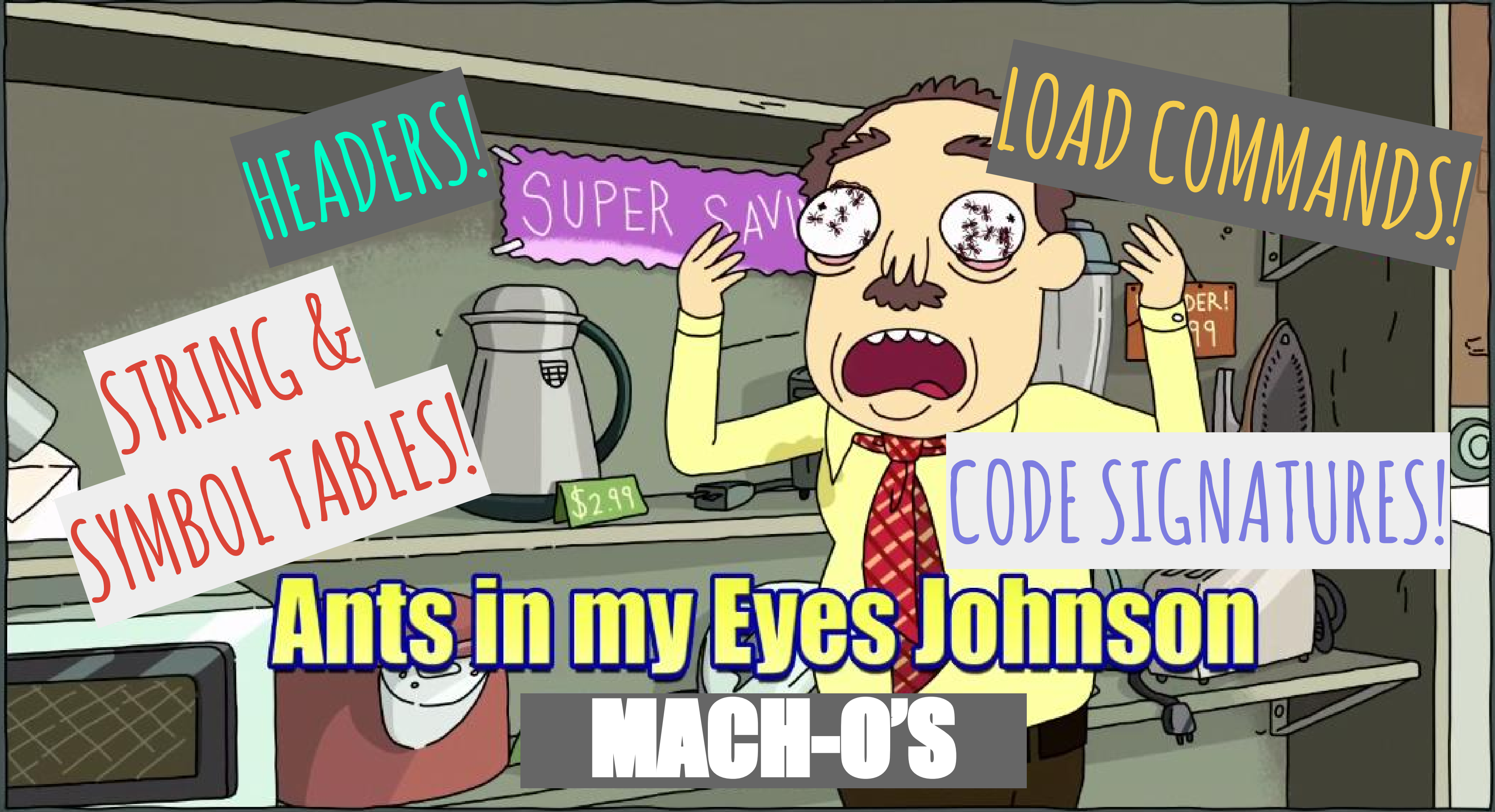
```
/*
 * C form of a CodeDirectory.
 */
typedef struct __CodeDirectory {
    uint32_t magic;
    uint32_t length;
    uint32_t version;
    uint32_t flags;
    uint32_t hashOffset;
    uint32_t identOffset;
    uint32_t nSpecialSlots;
    uint32_t nCodeSlots;
    uint32_t codeLimit;
    uint8_t  hashSize;
    uint8_t  hashType;
    uint8_t  spare1;
    uint8_t  pageSize;
    uint32_t spare2;
    /* followed by dynamic
       contentas located by
       offset fields above */
} CS_CodeDirectory;
```

Universal (FAT) Binaries



```
$ file /usr/bin/python
/usr/bin/python: Mach-O universal binary with 2 architectures
/usr/bin/python (for architecture i386):   Mach-O executable i386
/usr/bin/python (for architecture x86_64): Mach-O 64-bit executable x86_64

$ file /usr/lib/dyld
/usr/lib/dyld: Mach-O universal binary with 2 architectures
/usr/lib/dyld (for architecture x86_64):   Mach-O 64-bit dynamic linker x86_64
/usr/lib/dyld (for architecture i386):Mach-O dynamic linker i386

$ file /usr/bin/* | grep ‘universal’ | wc -l
    120
```

Yo binary so fat, its Mach-O’s got Mach-O’s!



HEADERS!

LOAD COMMANDS!

STRING & SYMBOL TABLES!

CODE SIGNATURES!

MACH-O'S

Additional Features

Convenience & Usability

Hashing
(md5, sha1, sha256)

File Entropy

Multiple input files

Output file

Abnormalities
(error handling,
work in progress)

Summon the Demo Demons

Hurdles & Lessons Learned

Documentation on the Mach-O format is sparse, and scattered across the interwebz, some of it pretty well hidden.

Reading other people's code sucks.

Just because it's not all human readable, doesn't mean it's not worth reading. The information is detailed, and potentially very useful.

Areas for Improvement

Code quality, consistency, robustness, etc.

Documentation (spelunking shouldn’t be a headache)

Error handling (understanding errors)

Moving Forward

PARSED ALL THE THINGS... NOW WHAT?

What can we learn from all this data?

How do we give it context and understand it?

How do we automate this process?

Finding Evil...

Dynamic Libraries

Functions/Classes/Methods

Strings

Abnormalities

Code Signature

Encryption (Good vs. Evil)

Toolchains

https://www.carbonblack.com/2016/03/01/analyzing-entrypoint-instruction-differences-in-mach-o-files-with-mpesm/

This was almost a talk about machine learning

Machine learning is hard.

We built a really cool model.

2 key problems:

1. Size / Diversity of available corpus
2. Training Set

Overfitting is a thing.

We're not giving up though.

If you're interested:
aaron@icebrg.io
will@icebrg.io

What's Next?

Future goals...

Continue to build corpus (big bucket 'o binaries)

Feature selection

Classification

Clustering

Malware discovery!

Questions?

HTTPS://GITHUB.COM/AARONST/MACHOLIBRE.GIT

AARON@ICEBRG.IO

WILL@ICEBRG.IO

WWW.ICEBRG.IO

THANK YOU!!!