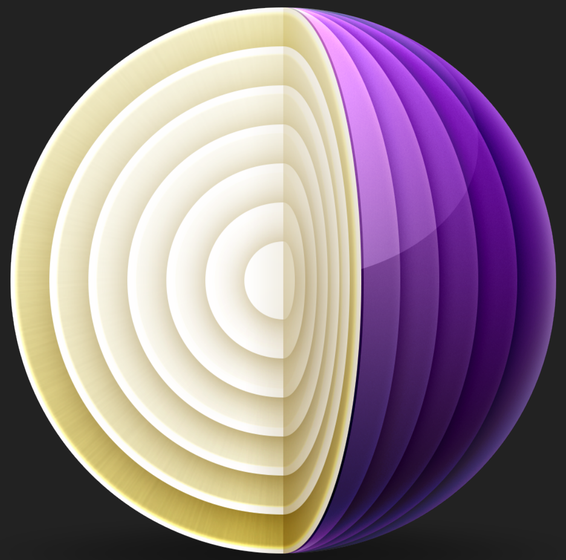
ITSECX -  @FREDERICJACOBS

REFLECTIONS ON TRUSTING

BITCODE

~ >>> WHOIS FREDERIC



iCepa

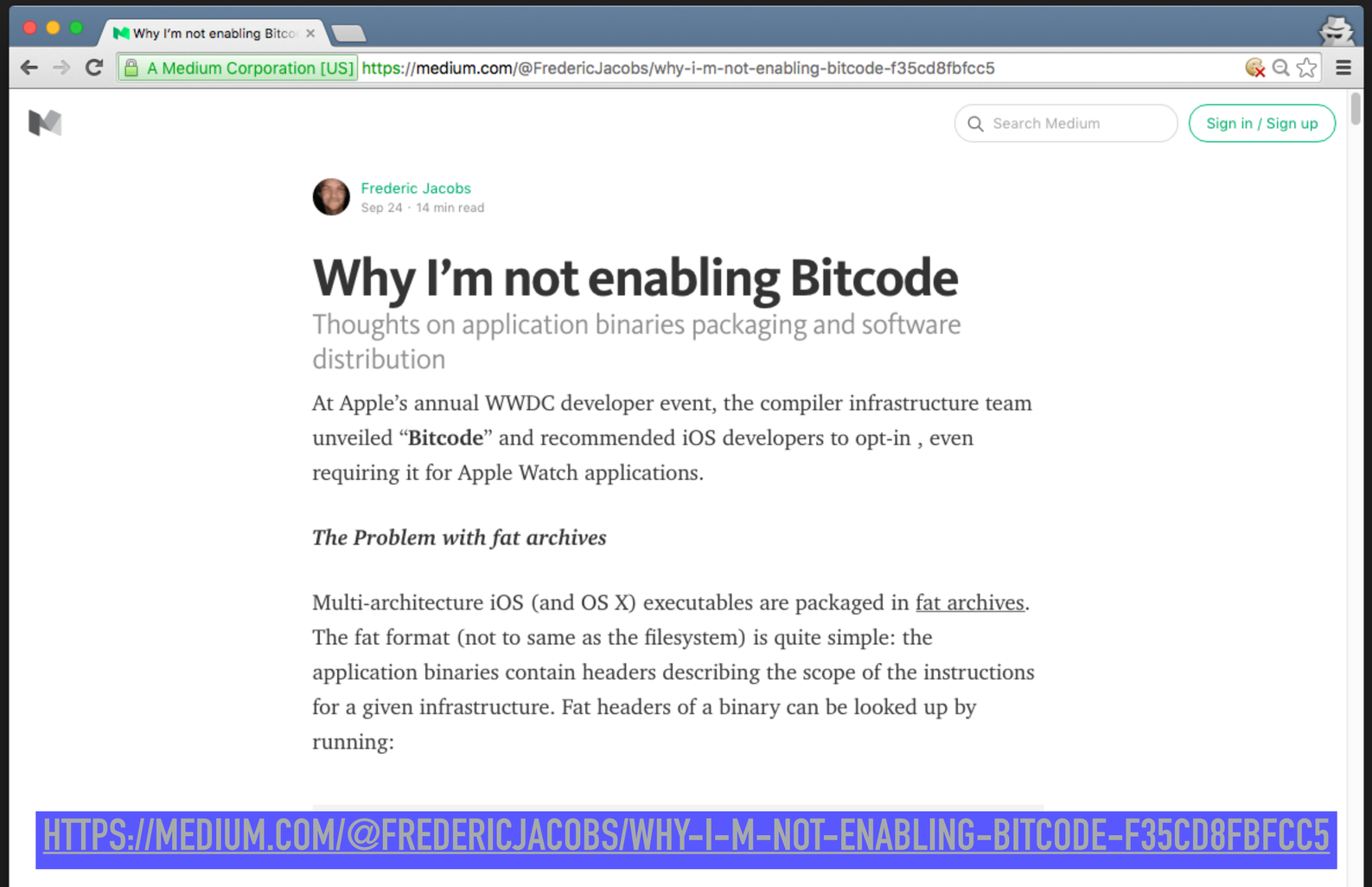
COMING SOON
Available on the App Store

Signal

Available on the App Store

Both Open-Source Software

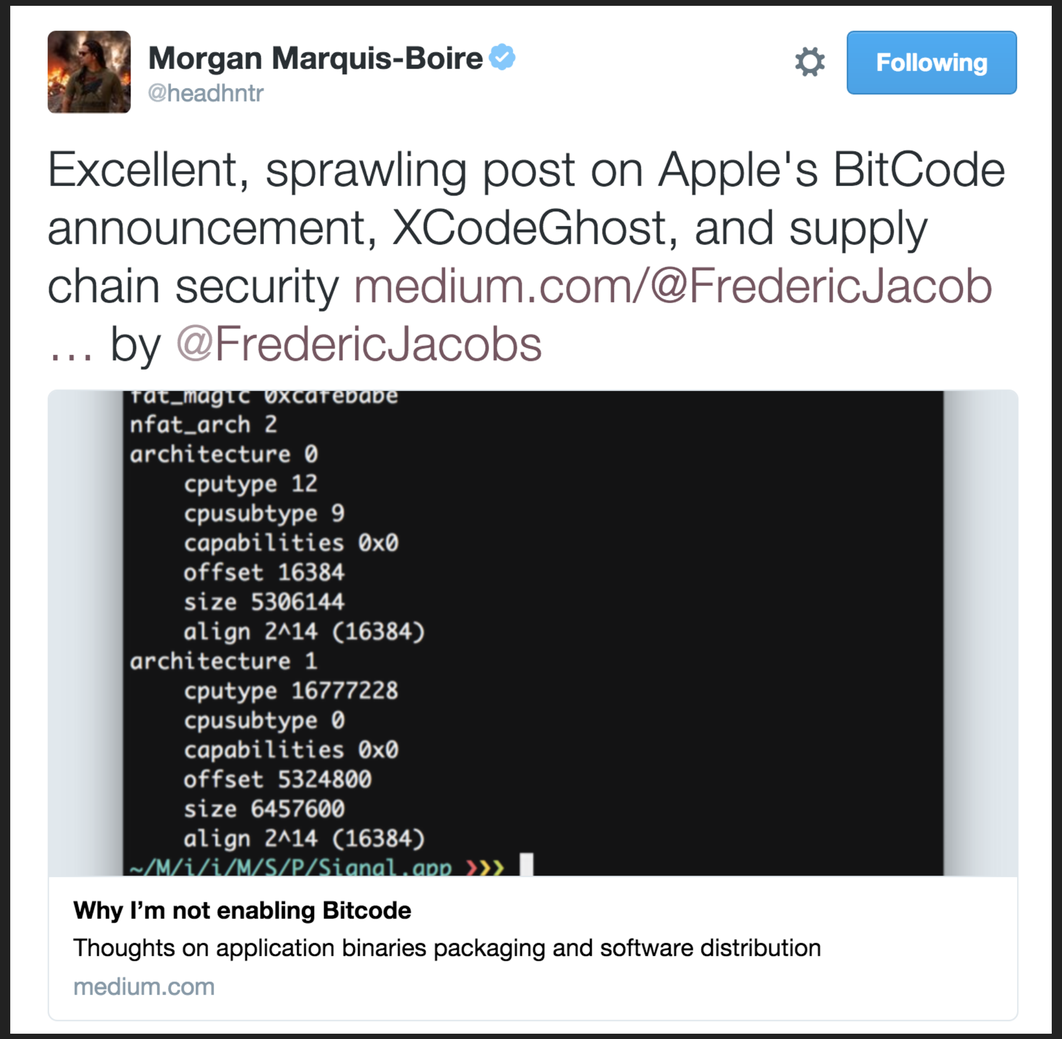
Talk based on blog post



MOTIVATION

Understanding the impact of Bitcode on the distribution of iOS security-sensitive software.

OR HOW MORGAN PUT IT



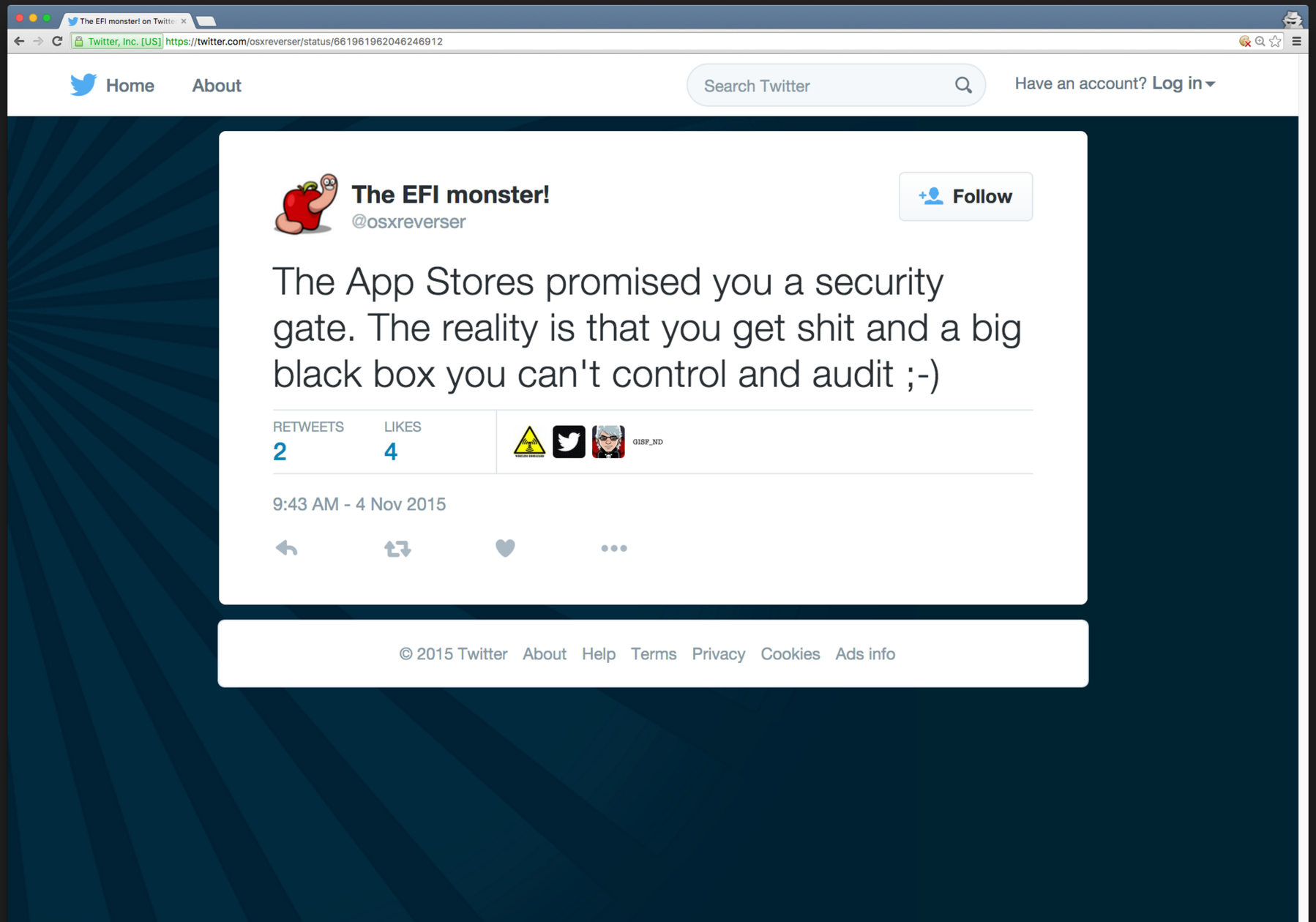
OUTLINE

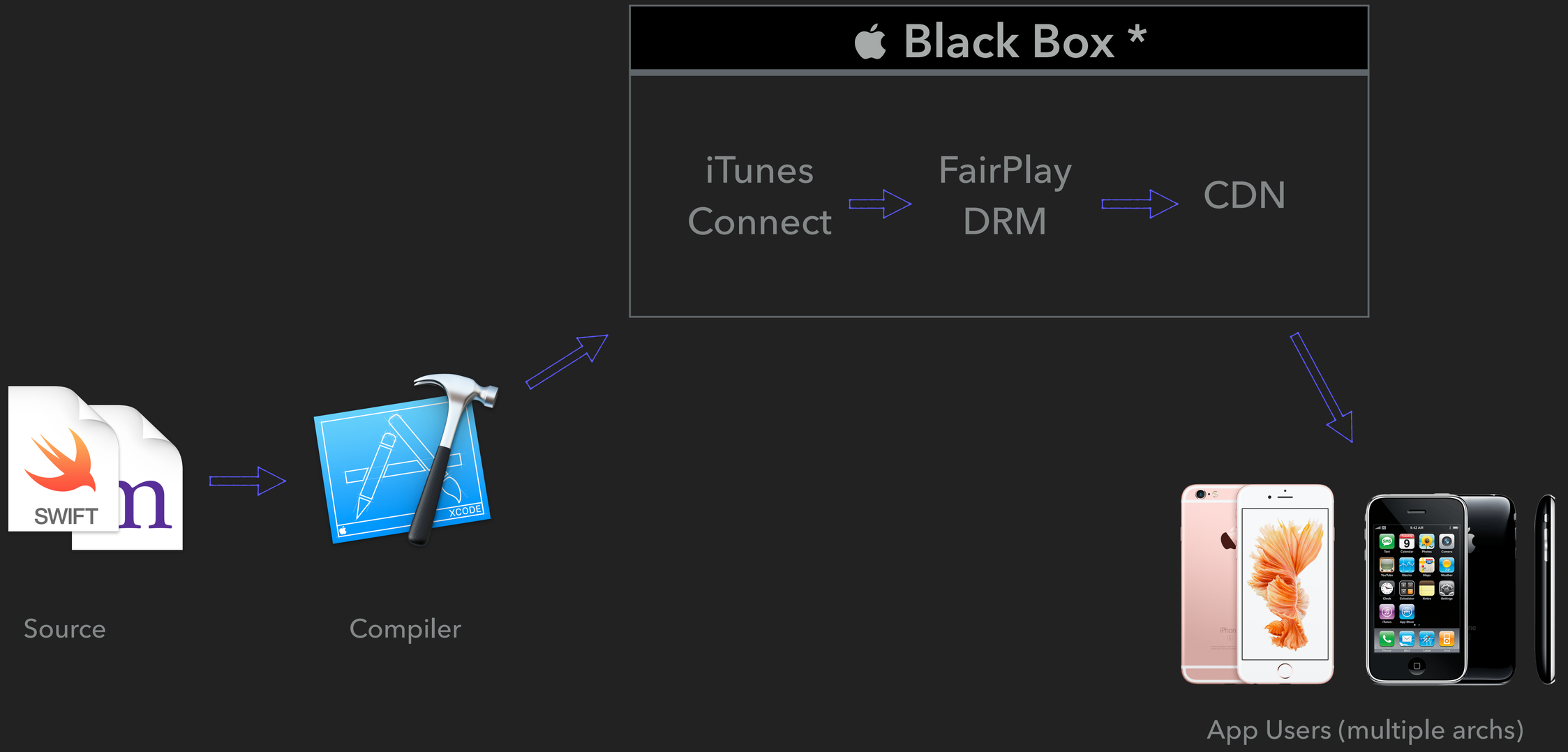
Software Distribution

Bitcode

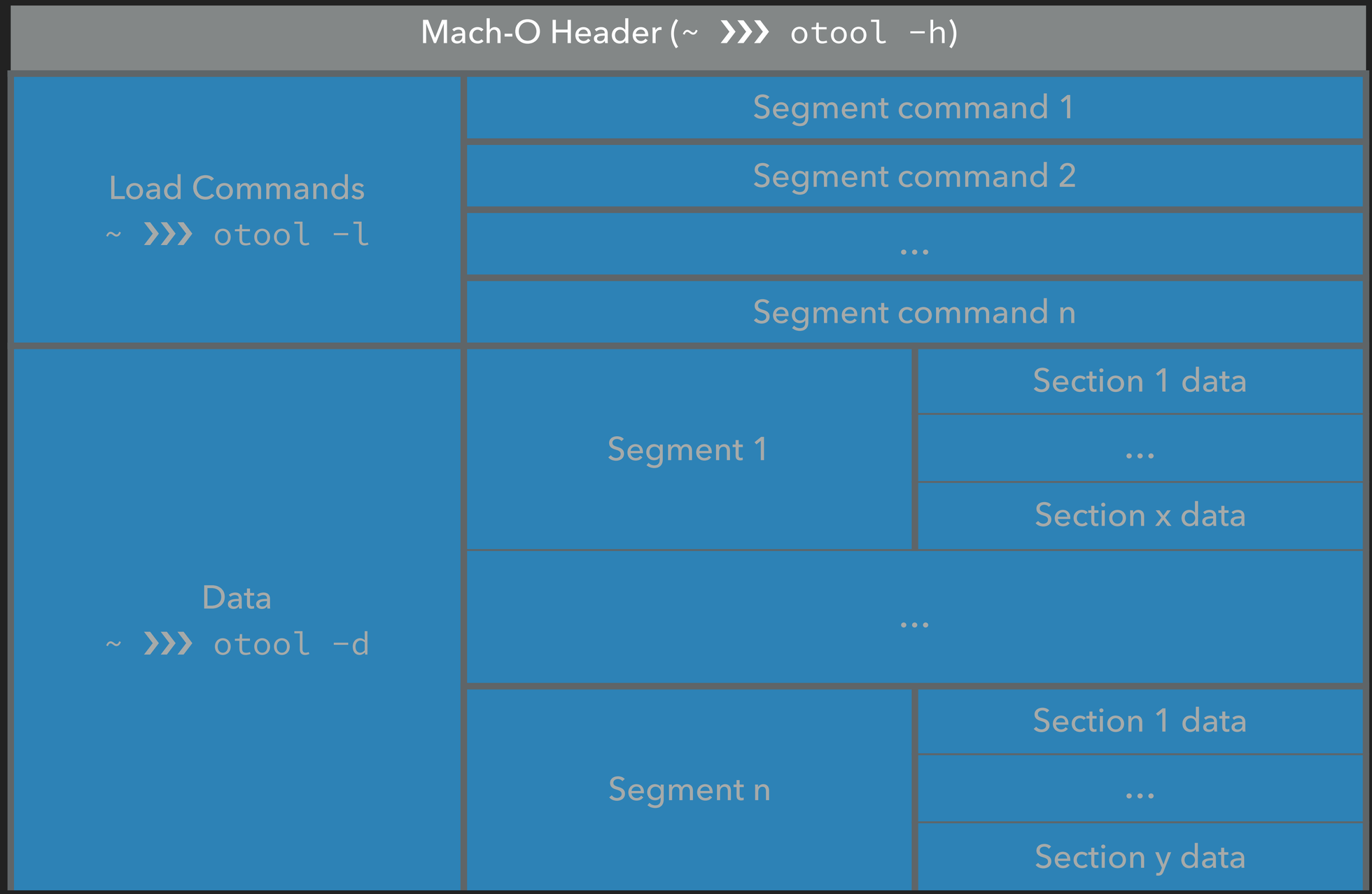
Security Implications

SOFTWARE DISTRIBUTION

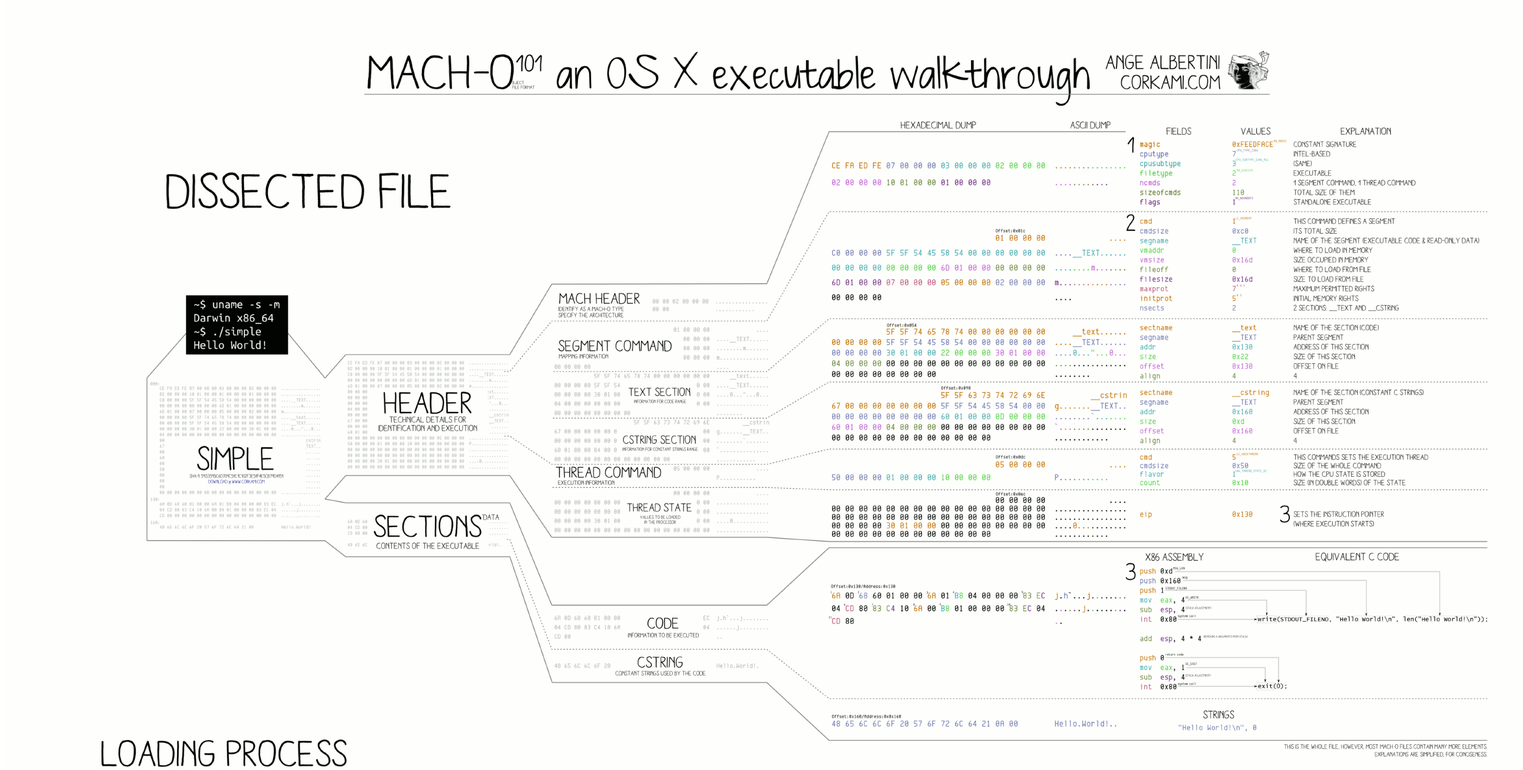

The EFI monster! on Twitter
Twitter, Inc. [US] https://twitter.com/osxreverser/status/661961962046246912
Home
About
Search Twitter
Have an account? Log in
The EFI monster!
@osxreverser
Follow
The App Stores promised you a security gate. The reality is that you get shit and a big black box you can't control and audit ;-)
RETWEETS
2
LIKES
4
GISF_ND
9:43 AM - 4 Nov 2015
© 2015 Twitter About Help Terms Privacy Cookies Ads info



* Heh, it's a black box, I actually have no clue what's really in there but given functionality these components must be there.


Mach-O Header (~ ››› otool -h)
Load Commands
~ ››› otool -l
Segment command 1
Segment command 2
...
Segment command n
Data
~ ››› otool -d
Segment 1
Section 1 data
...
Section x data
...
Segment n
Section 1 data
...
Section y data

Detailed Mach-O Description



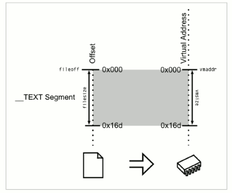
LOADING PROCESS

1 HEADER

THE MACH-O HEADER IS PARSED
THE COMMANDS ARE PARSED

2 MAPPING

THE FILE IS MAPPED IN MEMORY ACCORDING TO ITS SEGMENT(S) AND SECTIONS



3 EXECUTION

THE THREAD STATE IS LOADED IN THE CPU
EXECUTION IS RESUMED

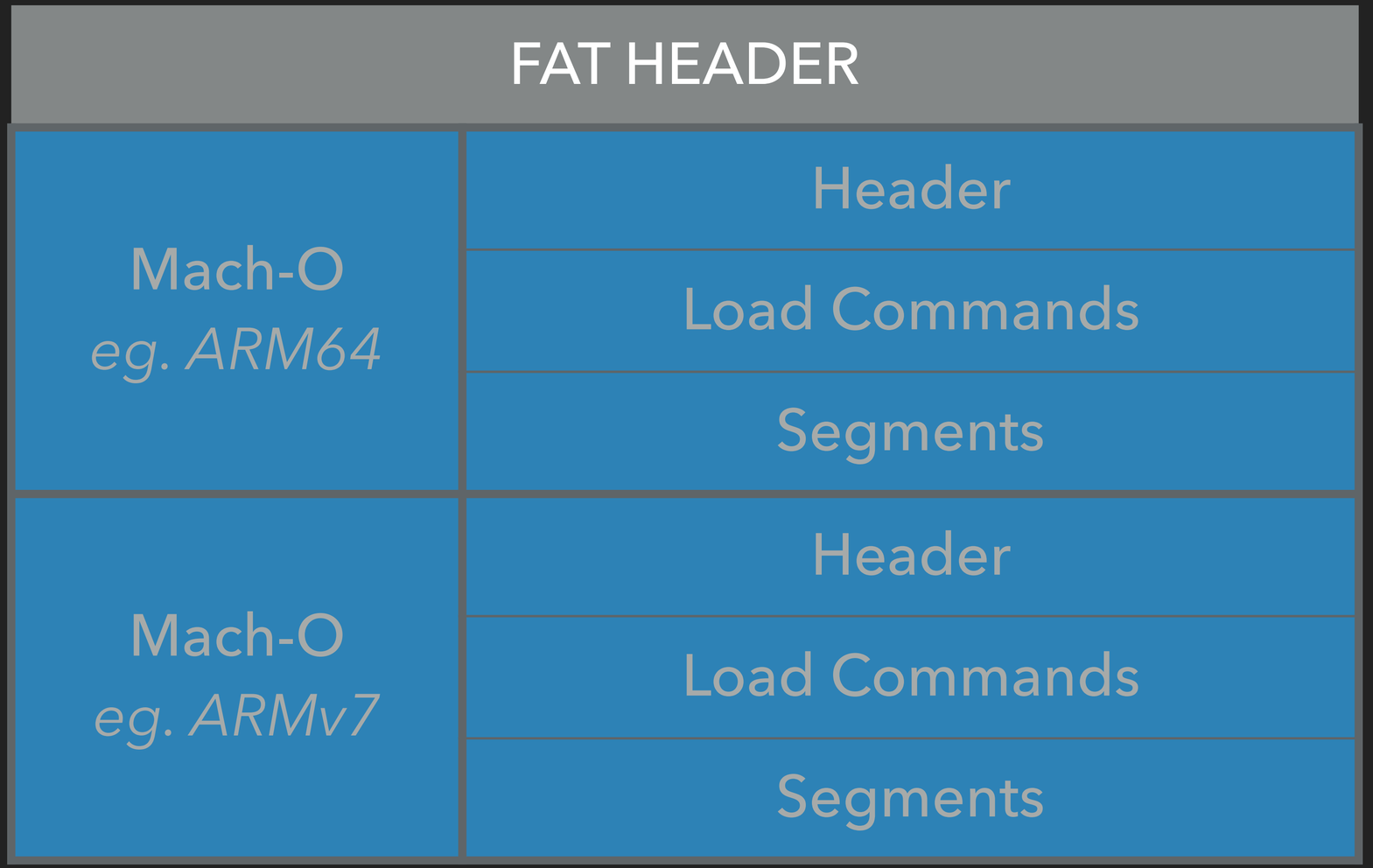
SYSCALLS ARE ACCESSED VIA:
- SYSCALL NUMBER IN THE EAX REGISTER
- CALLING INTERRUPT 0X80

TRIVIA

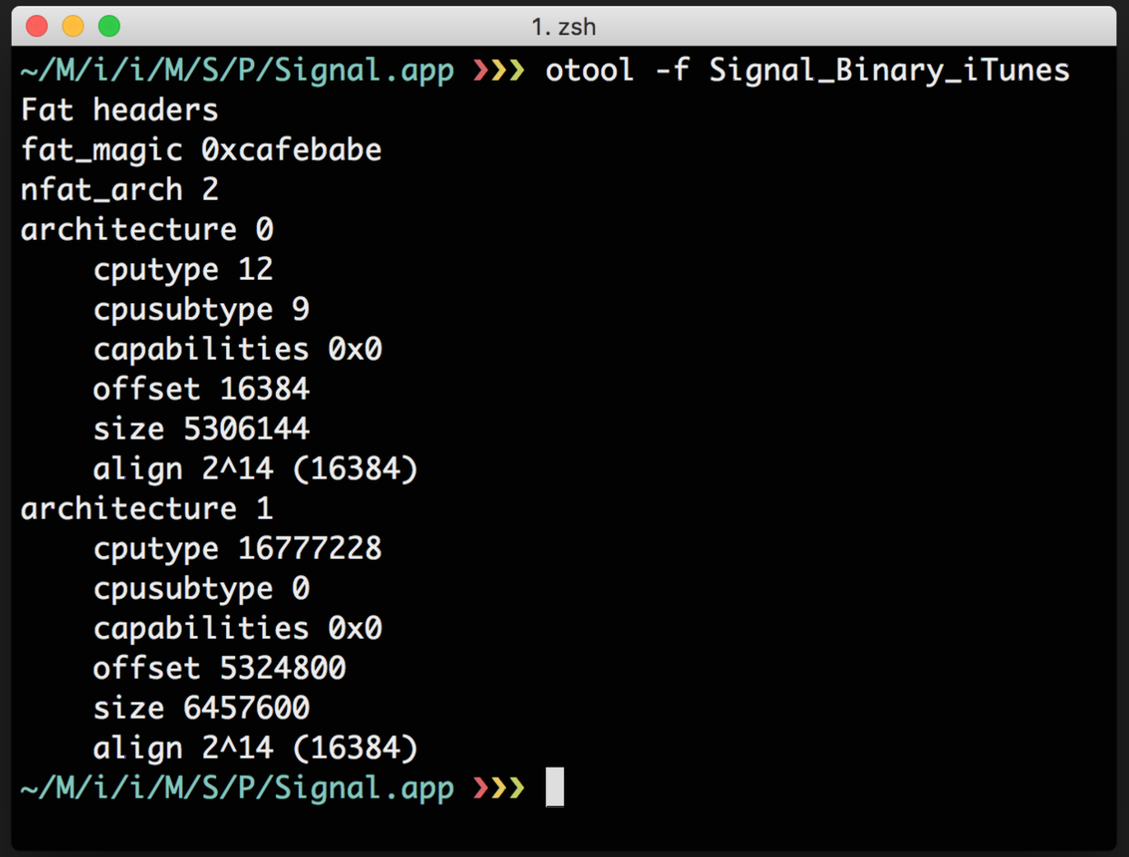
THE MACH-O FORMAT COMES FROM THE MACH KERNEL, CREATED AT C"M U" IN 1985

IT'S USED AMONG OTHERS,
BY OS X, IOS, "STEP..."
ON IPHONES, IPODS, MACS...

UNIVERSAL MACH-O FILES



FAT HEADERS LOOKUP



- nfat_arch: Number of supported architectures
- fat_magic: Defines configuration: 32 vs 64 bits, endianness
- armv7: ARM has CPU type 12, armv7 is CPU subtype 9. (The armv7-specific instructions start at offset 16384 and go on for the next 5306144 bytes.)
- arm64: CPU type 16777228, the ARM 64-bit CPU type.

iOS Support Matrix

Autumn 2014 Edition - v3.2.0
http://iossupportmatrix.com
@isupportmatrix
English

	ARMv6				ARMv7										ARMv7s			ARM64							
	iPhone June 2007	iPod touch September 2007	iPhone 3G July 2008	iPod touch (2nd gen) September 2008	iPhone 3GS June 2009	iPod touch (3rd gen) September 2009	iPad April 2010	iPhone 4 June 2010	iPod touch (4th gen) September 2010	iPad 2 March 2011	iPhone 4S October 2011	[new] iPad March 2012	iPod touch (5th gen) September 2012	iPad mini October 2012	iPhone 5 September 2012	iPad October 2012	iPhone 5c September 2013	iPhone 5s September 2013	iPad Air October 2013	iPad mini 2 October 2013	iPhone 6 September 2014	iPhone 6 Plus September 2014	iPad Air 2 October 2014	iPad mini 3 October 2014
Device Compatibility	128MB	128MB	128MB	128MB	256MB	256MB	A4 256MB	A4 512MB	A4 256MB	A5 512MB	A5 512MB	A5X 1GB	A5 512MB	A5 512MB	A6 1GB	A6X 1GB	A6 1GB	A7 1GB	A7 1GB	A7 1GB	A8 1GB	A8 1GB	A8X 2GB	A7 1GB
iPhone OS 1.0 Code name: Alpine, Heavenly, Little Bear, Snowbird, Oktoberfest	1.0	1.1																						
iPhone SDK 2.0 Code name: Big Bear, Sugarbowl, Timberline			2.0	2.1.1																				
iPhone SDK 3.0 Code name: Kirkwood, Northstar, Wildcat	136* 3.1.3	135* 3.1.3			3.0	3.1.1	iPad SDK 3.2																	
iPhone SDK 4.0 Code name: Apex, Baker, Jasper, Phoenix,Durango			136* 4.2.1	188* 4.2.1			4.3.5	(GSM) 4.0 / 4.2.6 (CDMA)	4.2.1	4.3.5														
iOS 5 Code name: Telluride, Hoodoo						278* 5.1.1	454* 5.1.1				5.0	5.1												
iOS 6 Code name: Sundance, Brighton					150/149 6.1.6				209/209 6.1.6				6.0	6.0	6.0	6.0								
iOS 7 Code name: Innsbruck, Sochi								206/206 7.1.2									7.0	7.0	7.0	7.0				
iOS 8 Code name: Okemo, OkemoTaos, OkemoZurs, Stowe, Copper										8.1	8.1	8.1	8.1	8.1	8.1	8.1	8.1	8.1	8.1	8.1	8.1	8.1	8.1	8.1
iOS 9 Code name: Monarch										262/495 9.0b	215/406 9.0b	260/493 9.0b	215/410 9.0b	260/491 9.0b	712/1278 9.0b	773/1408 9.0b	698/1249 9.0b	1363/2456 9.0b	1304/2528 9.0b	1418/2568 9.0b	1610/2880 9.0b	1596/2854 9.0b	1794/4441 9.0b	1418/2568 9.0b
Model ID	iPhone1,1	iPod1,1	iPhone1,2	iPod2,1	iPhone2,1	iPod3,1	iPad1,1	iPhone3,1	iPod4,1	iPad2,1 iPad2,4	iPhone4,1	iPad3,1 iPad3,2	iPod5,1	iPad2,5 iPad2,6	iPhone5,1 iPhone5,2	iPad3,4	iPhone5,3 iPhone5,4	iPhone6,1 iPhone6,2	iPad4,1 iPad4,2	iPad4,4 iPad4,5	iPhone7,2	iPhone7,1	iPad5,3 iPad5,4	iPad4,7 iPad4,8 iPad4,9

Key

A7 1GB — Chip generation / Device memory

- Accelerometer
- Bluetooth LE
- GPS (Cellular iPad only)
- Microphone
- Still Camera
- Retina Display
- ARM Version
- Camera Flash
- Gyroscope
- OpenGL ES Version
- Telephony
- Lightning Port
- M7/M8 Coprocessor
- Front Facing Camera
- Location Services
- Peer-to-Peer
- Video Camera
- TouchID
- Auto-Focus Camera
- Game Kit
- Magnetometer
- SMS
- Wifi
- Apple Pay

Do not support / Full support

4.2.1 — Earliest release / Latest release

150/149 — Geek Bench rating (Single Core/Multi Core)

* Geekbench v2 scores only

UNIVERSAL BINARIES

- Advantages
  - Supports multiple (micro-)architectures in a single file
  - User can back up older iPhone and restore on device with newer architecture without having to download any additional material
- Disadvantages
  - They are fat (yeah, they are literally called fat binaries)
    Adding new architectures comes at an important storage cost

iOS 9

App Thinning

- Goal: Reduce App Store downloads size
- How? Only distribute to the user necessary resources according to screen resolution and device architecture
- No recompilation

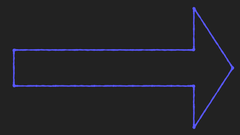
BITCODE

LLVM Frontend

```
~ ❯❯❯ cat helloworld.c

/* Hello World program */

#include<stdio.h>

main()
{
    printf("Hello World");
}

~ ❯❯❯ clang helloworld.c
```

LLVM-IR

Wait, why does it specify a target?



- LLVM IR isn't target independent unlike what is widely believed.
  - Clang source languages (ObjC, C, C++ ...) semantics are not target independent, so IR can't be target independent.

```
; ModuleID = 'helloworld.c'
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-
S128"
target triple = "x86_64-apple-macosx10.11.0"

@.str = private unnamed_addr constant [12 x i8] c"Hello
World\00", align 1

; Function Attrs: nounwind ssp uwtable
define i32 @main() #0 {
  %1 = call i32 (i8*, ...)* @printf(i8* getelementptr
inbounds ([12 x i8]* @.str, i32 0, i32 0))
  ret i32 0
}

declare i32 @printf(i8*, ...) #1

attributes #0 = { nounwind ssp uwtable "less-precise-
fpmad"="false" "no-frame-pointer-elim"="true" "no-
frame-pointer-elim-non-leaf" "no-infs-fp-math"="false"
"no-nans-fp-math"="false" "stack-protector-buffer-
size"="8" "target-cpu"="core2" "target-
features"="+ssse3,+cx16,+sse,+sse2,+sse3" "unsafe-fp-
math"="false" "use-soft-float"="false" }
attributes #1 = { "less-precise-fpmad"="false" "no-
frame-pointer-elim"="true" "no-frame-pointer-elim-non-
leaf" "no-infs-fp-math"="false" "no-nans-fp-
math"="false" "stack-protector-buffer-size"="8"
"target-cpu"="core2" "target-
features"="+ssse3,+cx16,+sse,+sse2,+sse3" "unsafe-fp-
math"="false" "use-soft-float"="false" }

!llvm.module.flags = !{!0}
!llvm.ident = !{!1}

!0 = !{i32 1, !"PIC Level", i32 2}
!1 = !{!"Apple LLVM version 7.0.0 (clang-700.1.76)"}
```

LLVM Backend

- Compiling LLVM to target binary
- Target optimized codegen
- Since IR depends on target, targets have to share the same or very similar ISA.

```
   .section  __TEXT,__text,regular,pure_instructions
   .macosx_version_min 10, 11
   .globl    _main
   .align    4, 0x90
_main:                                  ## @main
   .cfi_startproc
## BB#0:
   push  rbp
Ltmp0:
   .cfi_def_cfa_offset 16
Ltmp1:
   .cfi_offset rbp, -16
   mov   rbp, rsp
Ltmp2:
   .cfi_def_cfa_register rbp
   sub   rsp, 16
   lea   rdi, [rip + L_.str]
   mov   al, 0
   call  _printf
   xor   ecx, ecx
   mov   dword ptr [rbp - 4], eax ## 4-byte Spill
   mov   eax, ecx
   add   rsp, 16
   pop   rbp
   ret
   .cfi_endproc

   .section  __TEXT,__cstring,cstring_literals
L_.str:                                 ## @.str
   .asciz    "Hello World"


.subsections_via_symbols
```

Bitcode Recompilation

- Bitcode as a format is nothing new. It's a bitstream representation of the LLVM-IR
- Goal: Provide micro-architecture optimizations for new devices from day 1.
  - Eg. NEON floating-point or hardware integer division
- Apple asks developers to upload Bitcode along with their binary, recompiles for new (micro-)architectures assuming they have the same ISA
- Separate from App Thinning

Bitcode Binaries (as uploaded to iTC)

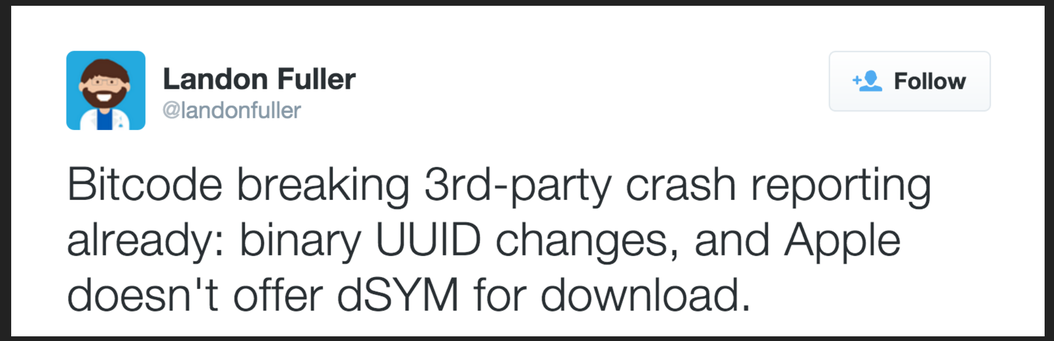
- Fat archives with multiple architectures
- For each architecture, contains a specific Mach-O Segment `__LLVM` containing a section `__bundle`.
  - The data segment contains a xar archive containing an LLVM Bitcode file per class

Extracting LLVM IR from Bitcode binaries

- bitcode_retreiver scans each architecture of the binary for the __LLVM segment and dumps it.
- Un-xar the Bitcode of the interested architecture (eg. `xar -x -f arm.xar`)
- Convert Bitcode Bitstream to human-readable LLVM-IR
- The LLVM IR can be obtained from the Bitcode by deserializing the bitstream with `llvm-dis`

Symbolication

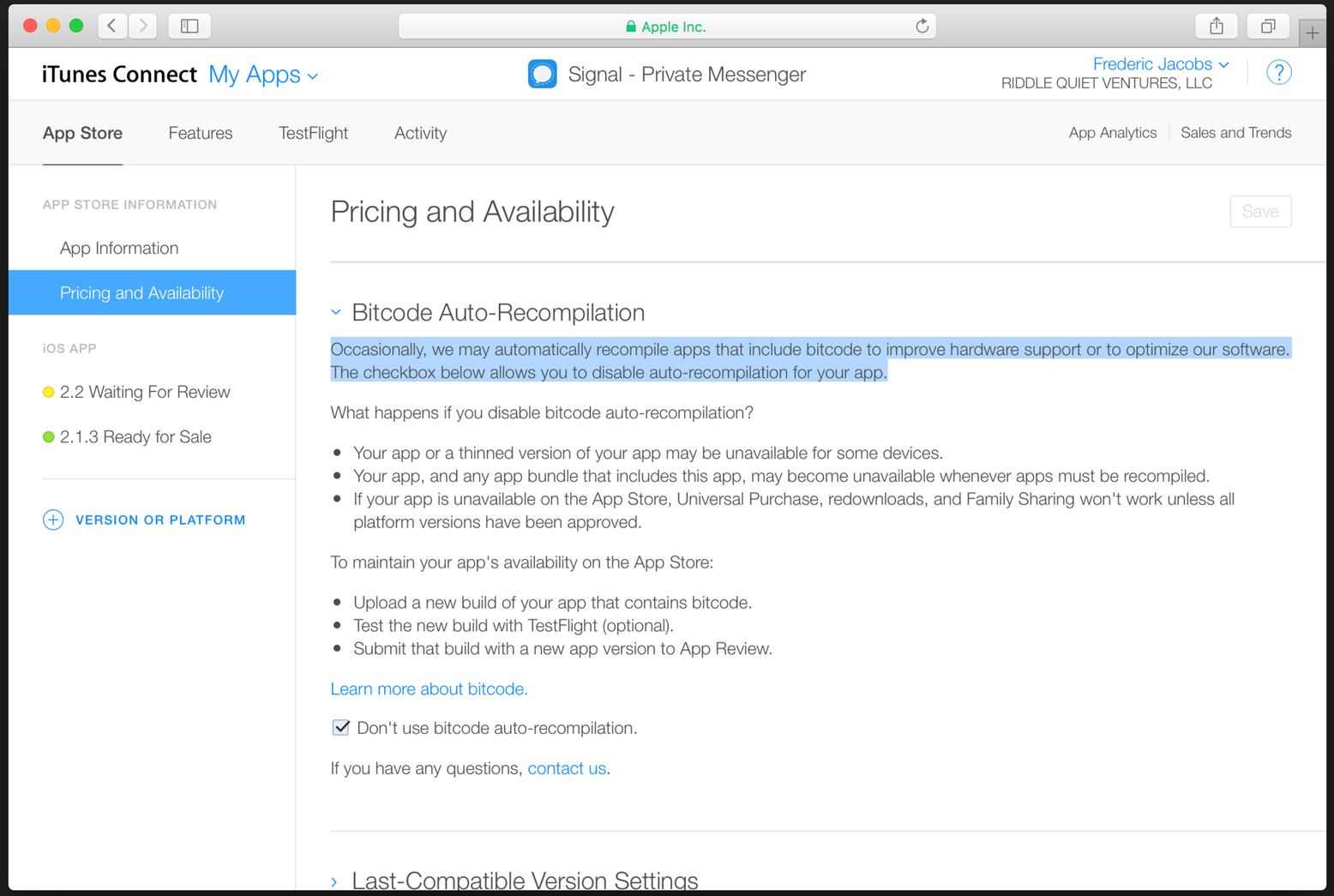
- Application submitted with
  - .dSYM for compiled targets
  - .bcsymbolmap in case the Bitcode is recompiled, Apple will then provide new .dSYM files if recompiled
- Originally broken on launch according to some developers, now seems to be fixed.



iTunes Connect Opt-Out

- Opt-out possible on iOS
- Mandatory for tvOS and WatchOS apps

- ⚠ Apple might turn optimizations on or off with no control of developer
- ⚠ No information is provided about LLVM version or backend compilation flags



SECURITY IMPLICATIONS

COMPILER OPTIMIZATIONS CORRECTNESS

Persistent State

- Data remains in memory beyond developer enforced boundaries
- Examples
  - Dead Store Elimination: store instruction is removed by the compiler because the result won't be read in subsequent instructions.
  - Function Call Inlining: Merges the stack frames of the caller and the callee. This results in changing the scope of variables stored on the function stack.
  - Code Motion: Reorders instructions and basic blocks in program based on their dependencies. Allows the compiler to reorder instructions and basic blocks in the program based on their dependencies

Dead Store Elimination

```
#include <string>
using std::string;

#include <memory>

// The specifics of this function are
// not important for demonstrating this bug.
const string getPasswordFromUser() const;

bool isPasswordCorrect() {
    bool isPasswordCorrect = false;
    string Password("password");

    if(Password == getPasswordFromUser()) {
        isPasswordCorrect = true;
    }

    // This line is removed from the optimized code
    // even though it secures the code by wiping
    // the password from memory.
    memset(Password, 0, sizeof(Password));

    return isPasswordCorrect;
}
```

https://gcc.gnu.org/bugzilla/show_bug.cgi?id=8537

Function Inlining

```
string decrypt(cipherText) {
// code fetches decryption key
// and decrypts message in secure environment
}

// Fetches a message from the network
// Deserialize the packet
// Decrypt & parse result

void nextMessage() {
// Code in this function does not assume
// a trusted execution environment.
...
//call secure function
protoBuf = decrypt(msg);
...
}
```

Undefined Behavior

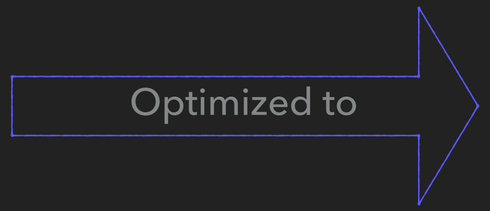
- Programming languages specifications intentionally leaving out semantic specifics for some operation
- Gives more freedom to implementors for optimizations and adapt to hardware
- GCC/Clang sometimes generate warnings for undefined behavior
- Examples in C:
  - uninitialized variable
  - dividing by zero
  - …

Side Channels

- Any attack based on information gained from the physical implementation of a cryptosystem, rather than brute force or theoretical weaknesses in the algorithms
- Time and energy consumption are common and exploitable side-channels
- To prevent an attacker to guess what code path was taken, never branch on secret material

Example: Common Subexpression Elimination

```
int crypt(int k*){
int key = 0;
if (k[0]==0xC0DE){
 key=k[0]*15+3;
 key+=k[1]*15+3;
 key+=k[2]*15+3;
} else {
 key=2*15+3;
 key+=2*15+3;
 key+=2*15+3;
}
```




```
int crypt(int k*){
int key = 0;
if (k[0]==0xC0DE){
 key=k[0]*15+3;
 key+=k[1]*15+3;
 key+=k[2]*15+3;
} else {
 tmp = 2*15+3;
 key = 3*tmp;
}
```

⚠️Timing attack: One code path has less instructions than the other

NO INLINE ASSEMBLY

There's just no way you can reasonably expect even the most advanced C compilers to do this on your behalf.

Mike Pall (LuaJIT author)
on possible optimizations

Inline Assembly

- Why?
  - Gives more tuning power to developer to prevent side channel leakage
  - Performance
- Widely used in cryptographic implementations (OpenSSL, Apple's own corecrypto ...)
  - But not only, VLC also uses inline assembly for optimizations

COMPILER TRUST

How Much Do You Trust Your Compiler?

- Compromised compiler/toolchain
  - XcodeGhost, distribution of backdoor compiler
- Control over dependencies
  - A single “CocoaPod” module can compromise entire app
  - Relying on bad library
- Compiler bugs
- Buggy compiler/environment

- With Bitcode specifically, incentives for an attacker to compromise re-compilation is huge. Exploit in build system could result in altering a huge amount of apps in App Store.

POTENTIAL FOR BACKDOOR INJECTION

Obama Administration Working Group discussing using automatic updates to insert backdoors

Provider-enabled remote access to encrypted devices through current update procedures. Virtually all consumer devices include the capability to remotely download and install updates to their operating system and applications. For this approach, law enforcement would use lawful process to compel providers to use their remote update capability to insert law enforcement software into a targeted device. Once inserted, such software could enable far-reaching access to and control of the targeted device. This proposal would not require physical modification of devices, and so would likely be less costly for providers to implement. It would also enable remote access, and make surreptitious access much less costly. However, its use could call into question the trustworthiness of established software update channels. Individual users, concerned about remote access to their devices, could choose to turn off software updates, rendering their devices significantly less secure as time passed and vulnerabilities were discovered by not patched.

REPRODUCIBLE BUILDS

Reproducible Builds Without Bitcode

- Verifying that a build on the App Store matches the code on GitHub is hard
  - Mostly due to FairPlay, Apple's FairPlay DRM
    - Can't opt out
  - Requires jailbroken iPhone, use lldb to dump binary at runtime

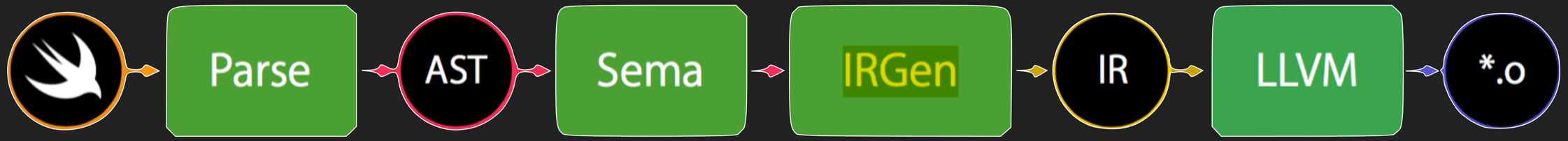
Reproducible Builds With Bitcode

- Same procedure for known architectures and if binaries were not recompiled
- But what happens when binaries don’t match? Cost of reverse engineering differences is important if you don’t have the toolchain

TOWARDS TARGET INDEPENDENCE?

Swift Intermediate Language

- Presented early November 2015 at the LLVM Dev Summit
- Higher-level IR
- Reducing undefined behavior
- IRGen handles type lowering & concretization of the target



QUESTIONS?

REFERENCES

- Bitcode Demystified by Alex Denisov
- Excellent resource on the Correctness-Security Gap from LangSec by Vijay D'Silva, Mathias Payer, Dawn Song
- Undefined behavior:
  - Undefined Behavior: What Happened to My Code?
  - http://blog.llvm.org/2011/05/what-every-c-programmer-should-know.html
  - http://blog.regehr.org/archives/213
- See clickable references in slides