Пишем свой iOS симулятор

Терминология

Virtualisation
Emulation

Виртуализация

Архитектура хоста

x86 → x86
arm → arm

Можно создать
“виртуальную машину”

Windows arm64 на Apple Silicon

Виртуализация

VirtualBox 

Parallels 

VMware Fusion 

Virtualization.framework 

Эмуляция

Любая архитектура

x86, arm, MIPS, RISC, …

Установка Linux x86 на Arm

Запуск NES игр под x86

Эмуляция

QEMU

UTM

PPSSPP
MIPS

Rosetta (v1/2)
Транслятор

iOS Simulator

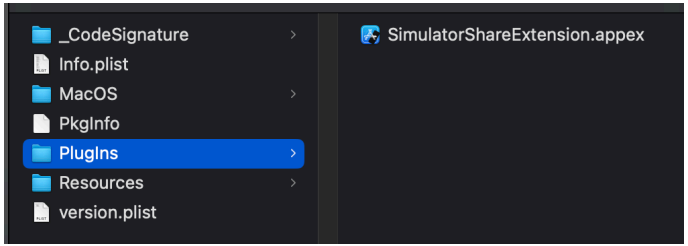
iOS Simulator

iOS Simulator

Очевидные факты

Приложение для macOS

Называется симулятор,
но не эмулятор



Умеет запускать специальные
приложения

iOS Simulator

На маках с процессорами intel — x86

Поддерживает только бинари своей архитектуры

На маках с процессорами M* — arm64

Виртуализация, но с искуственной ОС

Что мы будем делать?

Цели супер кратко

*Запустить** приложение для
iOS на macOS

Не использовать при этом iOS
Simulator

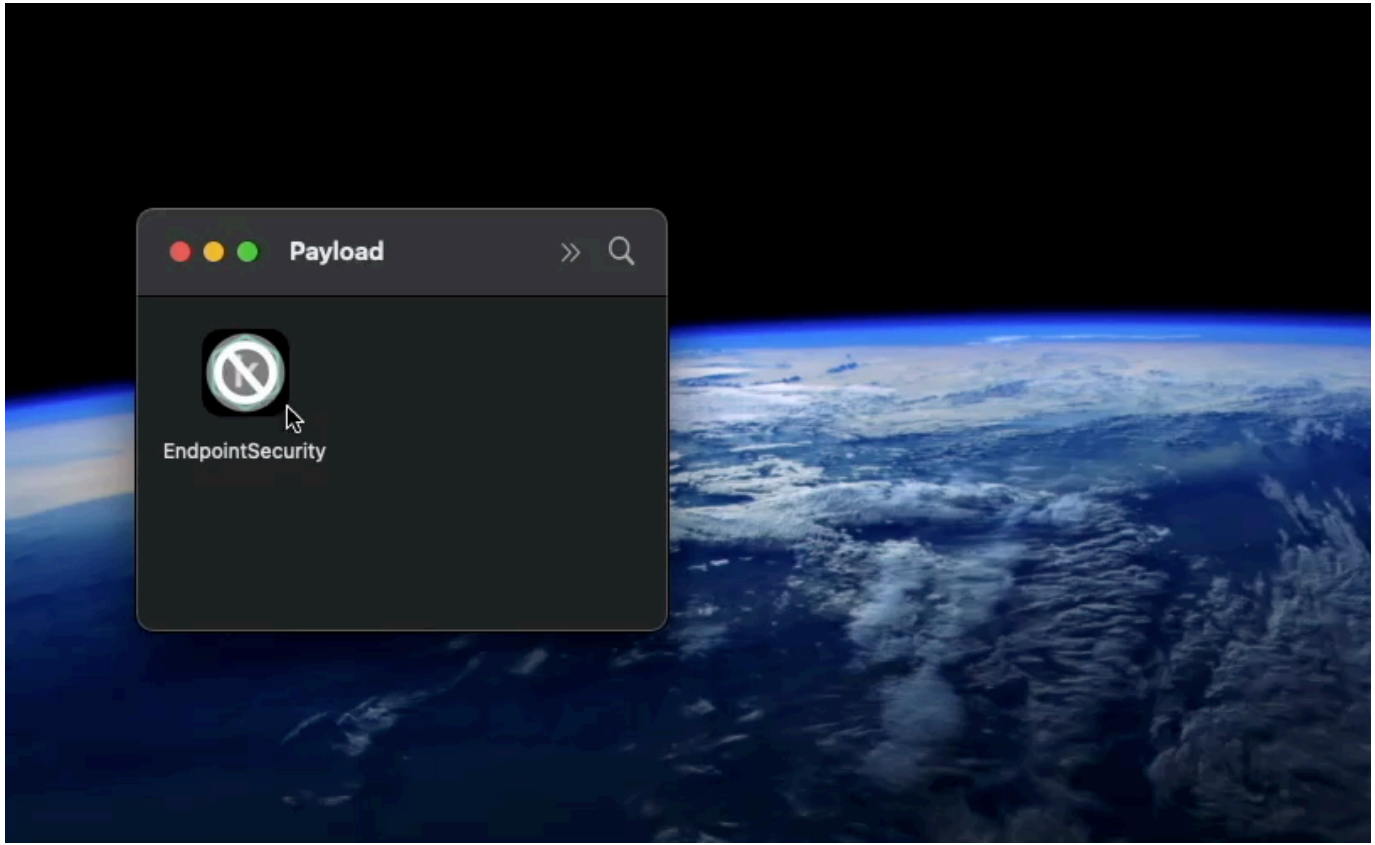
Какой план?

Попробуем запустить .app iOS на macOS as is

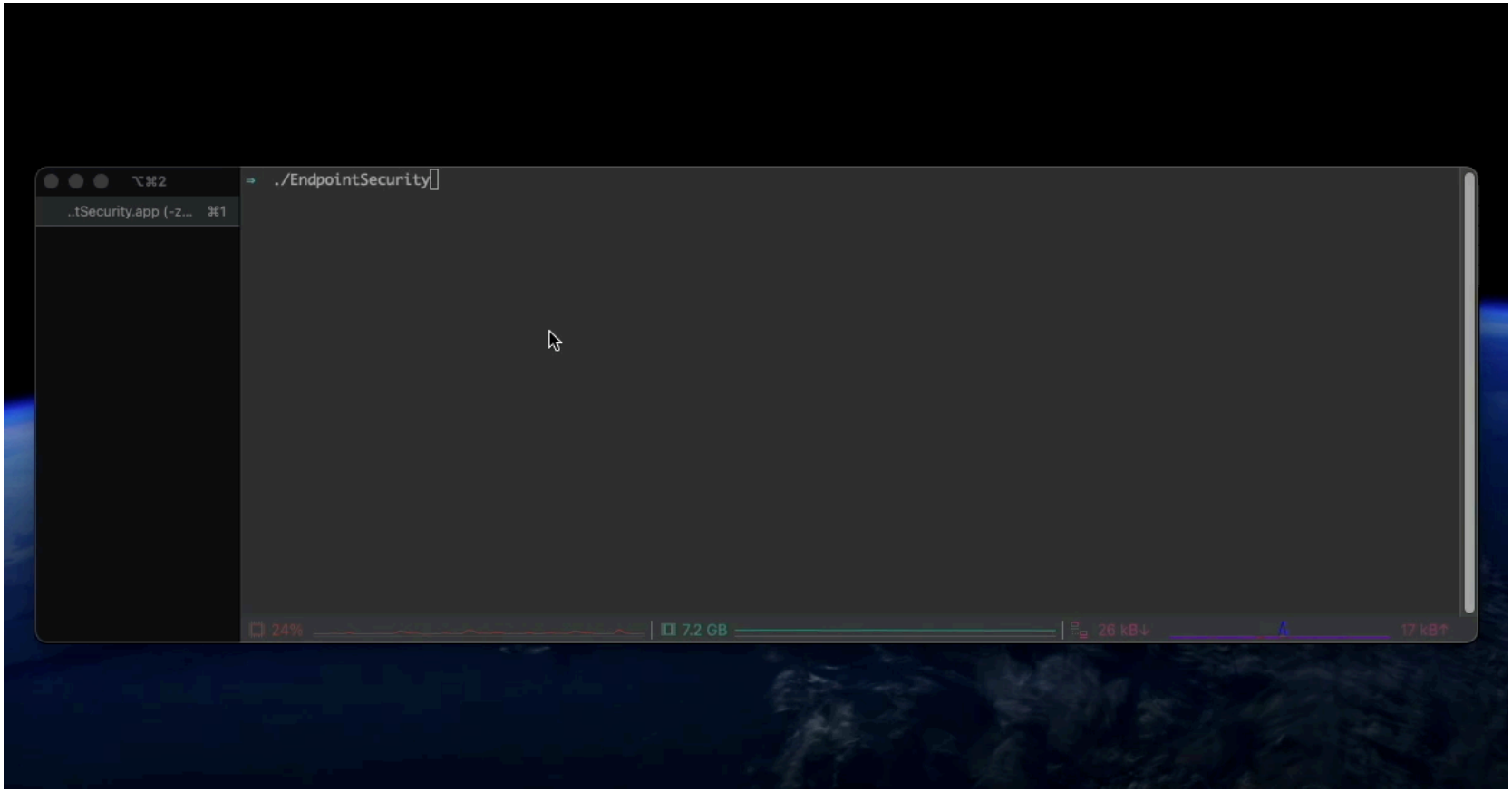
Разберёмся что происходит во время запуска

Попробуем запустить Mach-O сами

Запускаем .app на macOS


Payload
EndpointSecurity

Запускаем .app на macOS из консоли


⌥⌘2
..tSecurity.app (~z... ⌘1
./EndpointSecurity
24%
7.2 GB
26 kB↓
17 kB↑

“Давайте разбираться”

macOS может на многих этапах
отказываться запустить
бинарник

Этап 0 (.app)

смотрим в plist

```
plistutil -p ./Info.plist


"CFBundleSupportedPlatforms": [
    "iPhoneOS"
],
```

“Давайте разбираться”

macOS может на многих этапах отказываться запустить бинарник

Этап 0 (.app)

смотрим в plist

Этап 1 (из консоли)

проверяем платформу в Mach-O

Платформа

В одном из “мета” разделов Mach-O есть информация о платформе

0x2 Соответствует платформе iOS

В нашем бинаре указана платформа 0x2

macOS не будет такое запускать

Как написать iOS Simulator?

Понять, нужен нам симулятор, эмулятор, или виртуальная машина

Изучить строение бинаря iOS приложения

Разобраться в строении iOS app

Загрузить в память бинарь для исполнения

Как написать iOS Simulator?

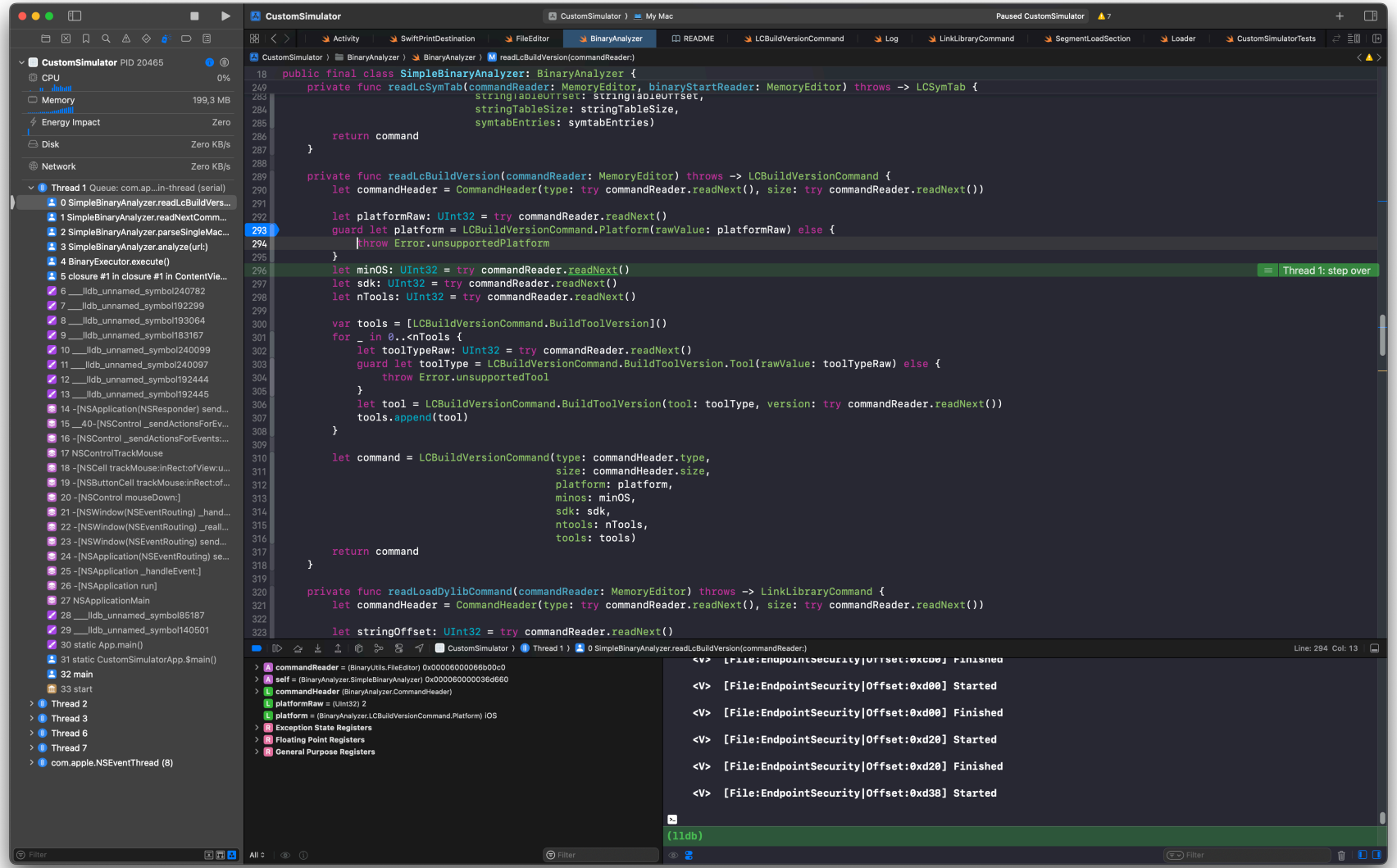
Разобраться с прилинкованными библиотеками

Запустить приложение, и радоваться

Инструменты

Какими инструментами будем пользоваться?

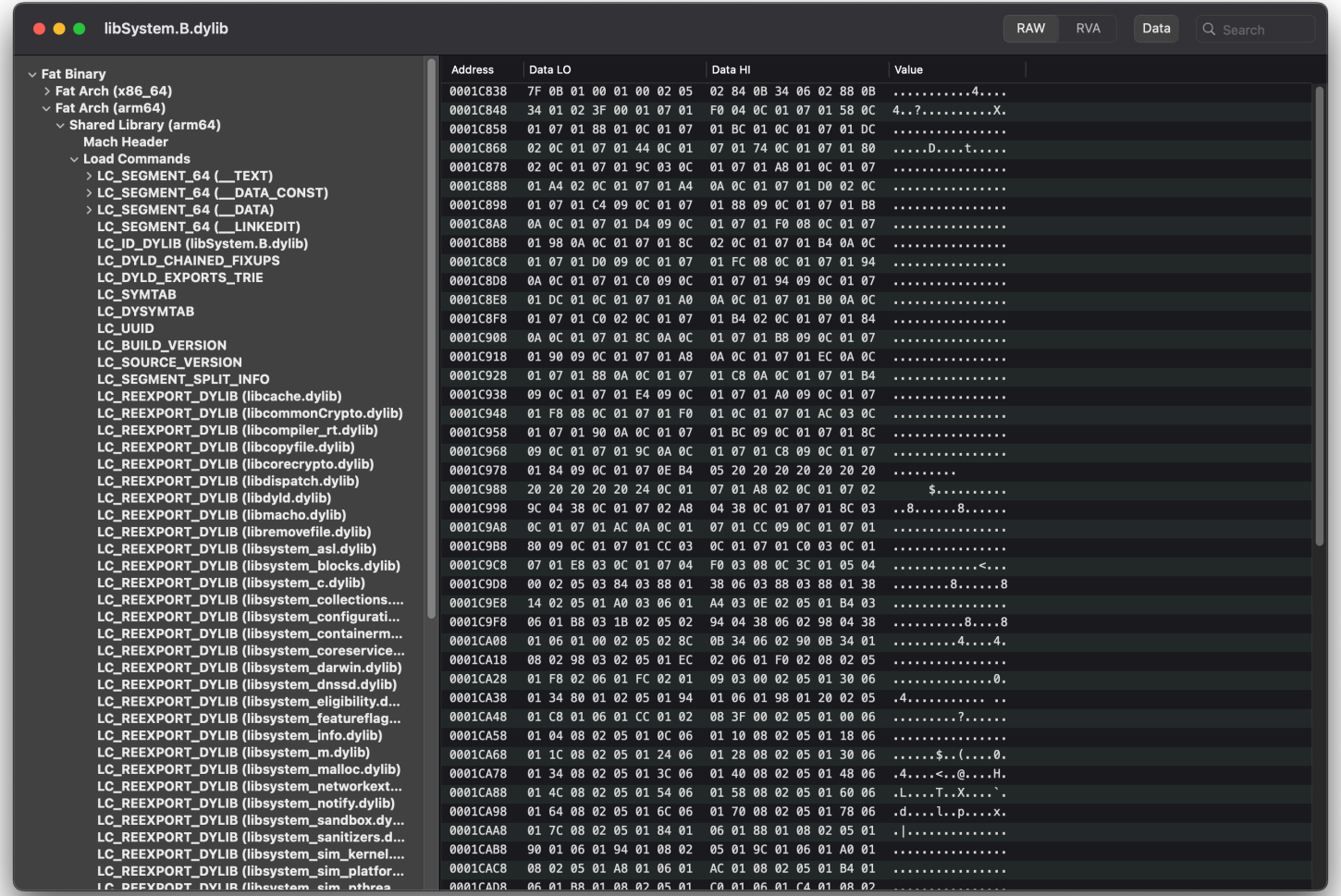
Xcode


CustomSimulator
CustomSimulator PID 20465
CPU 0%
Memory 199,3 MB
Energy Impact Zero
Disk Zero KB/s
Network Zero KB/s
Thread 1 Queue: com.ap...in-thread (serial)
0 SimpleBinaryAnalyzer.readLcBuildVers...
1 SimpleBinaryAnalyzer.readNextComm...
2 SimpleBinaryAnalyzer.parseSingleMac...
3 SimpleBinaryAnalyzer.analyze(url:)
4 BinaryExecutor.execute()
5 closure #1 in closure #1 in ContentVie...
Thread 2
Thread 3
Thread 6
Thread 7
com.apple.NSEventThread (8)
Activity
SwiftPrintDestination
FileEditor
BinaryAnalyzer
README
LCBuildVersionCommand
Log
LinkLibraryCommand
SegmentLoadSection
Loader
CustomSimulatorTests
public final class SimpleBinaryAnalyzer: BinaryAnalyzer {
private func readLcSymTab(commandReader: MemoryEditor, binaryStartReader: MemoryEditor) throws -> LCSymTab {
stringTableOffset: stringTableOffset,
stringTableSize: stringTableSize,
symtabEntries: symtabEntries)
return command
}
private func readLcBuildVersion(commandReader: MemoryEditor) throws -> LCBuildVersionCommand {
let commandHeader = CommandHeader(type: try commandReader.readNext(), size: try commandReader.readNext())
let platformRaw: UInt32 = try commandReader.readNext()
guard let platform = LCBuildVersionCommand.Platform(rawValue: platformRaw) else {
throw Error.unsupportedPlatform
}
let minOS: UInt32 = try commandReader.readNext()
Thread 1: step over
let sdk: UInt32 = try commandReader.readNext()
let nTools: UInt32 = try commandReader.readNext()
var tools = [LCBuildVersionCommand.BuildToolVersion]()
for _ in 0..<nTools {
let toolTypeRaw: UInt32 = try commandReader.readNext()
guard let toolType = LCBuildVersionCommand.BuildToolVersion.Tool(rawValue: toolTypeRaw) else {
throw Error.unsupportedTool
}
let tool = LCBuildVersionCommand.BuildToolVersion(tool: toolType, version: try commandReader.readNext())
tools.append(tool)
}
let command = LCBuildVersionCommand(type: commandHeader.type,
size: commandHeader.size,
platform: platform,
minos: minOS,
sdk: sdk,
ntools: nTools,
tools: tools)
return command
}
private func readLoadDylibCommand(commandReader: MemoryEditor) throws -> LinkLibraryCommand {
let commandHeader = CommandHeader(type: try commandReader.readNext(), size: try commandReader.readNext())
let stringOffset: UInt32 = try commandReader.readNext()
Line: 294 Col: 13
commandReader = (BinaryUtils.FileEditor) 0x00006000066b00c0
self = (BinaryAnalyzer.SimpleBinaryAnalyzer) 0x000060000036d660
commandHeader (BinaryAnalyzer.CommandHeader)
platformRaw = (UInt32) 2
platform = (BinaryAnalyzer.LCBuildVersionCommand.Platform) iOS
Exception State Registers
Floating Point Registers
General Purpose Registers
<V> [File:EndpointSecurity|Offset:0xd00] Started
<V> [File:EndpointSecurity|Offset:0xd00] Finished
<V> [File:EndpointSecurity|Offset:0xd20] Started
<V> [File:EndpointSecurity|Offset:0xd20] Finished
<V> [File:EndpointSecurity|Offset:0xd38] Started
(lldb)

Какими инструментами будем пользоваться?

Xcode

Различные Mach-O
анализаторы


libSystem.B.dylib
RAW
RVA
Data
Search
Fat Binary
Fat Arch (x86_64)
Fat Arch (arm64)
Shared Library (arm64)
Mach Header
Load Commands
LC_SEGMENT_64 (__TEXT)
LC_SEGMENT_64 (__DATA_CONST)
LC_SEGMENT_64 (__DATA)
LC_SEGMENT_64 (__LINKEDIT)
LC_ID_DYLIB (libSystem.B.dylib)
LC_DYLD_CHAINED_FIXUPS
LC_DYLD_EXPORTS_TRIE
LC_SYMTAB
LC_DYSYMTAB
LC_UUID
LC_BUILD_VERSION
LC_SOURCE_VERSION
LC_SEGMENT_SPLIT_INFO
LC_REEXPORT_DYLIB (libcache.dylib)
LC_REEXPORT_DYLIB (libcommonCrypto.dylib)
LC_REEXPORT_DYLIB (libcompiler_rt.dylib)
LC_REEXPORT_DYLIB (libcopyfile.dylib)
LC_REEXPORT_DYLIB (libcorecrypto.dylib)
LC_REEXPORT_DYLIB (libdispatch.dylib)
LC_REEXPORT_DYLIB (libdyld.dylib)
LC_REEXPORT_DYLIB (libmacho.dylib)
LC_REEXPORT_DYLIB (libremovefile.dylib)
LC_REEXPORT_DYLIB (libsystem_asl.dylib)
LC_REEXPORT_DYLIB (libsystem_blocks.dylib)
LC_REEXPORT_DYLIB (libsystem_c.dylib)
LC_REEXPORT_DYLIB (libsystem_collections....
LC_REEXPORT_DYLIB (libsystem_configurati...
LC_REEXPORT_DYLIB (libsystem_containerm...
LC_REEXPORT_DYLIB (libsystem_coreservice...
LC_REEXPORT_DYLIB (libsystem_darwin.dylib)
LC_REEXPORT_DYLIB (libsystem_dnssd.dylib)
LC_REEXPORT_DYLIB (libsystem_eligibility.d...
LC_REEXPORT_DYLIB (libsystem_featureflag...
LC_REEXPORT_DYLIB (libsystem_info.dylib)
LC_REEXPORT_DYLIB (libsystem_m.dylib)
LC_REEXPORT_DYLIB (libsystem_malloc.dylib)
LC_REEXPORT_DYLIB (libsystem_networkext...
LC_REEXPORT_DYLIB (libsystem_notify.dylib)
LC_REEXPORT_DYLIB (libsystem_sandbox.dy...
LC_REEXPORT_DYLIB (libsystem_sanitizers.d...
LC_REEXPORT_DYLIB (libsystem_sim_kernel....
LC_REEXPORT_DYLIB (libsystem_sim_platfor...
Address
Data LO
Data HI
Value
0001C838 7F 0B 01 00 01 00 02 05 02 84 0B 34 06 02 88 0B .....4....
0001C848 34 01 02 3F 00 01 07 01 F0 04 0C 01 07 01 58 0C 4..?.....X.
0001C858 01 07 01 88 01 0C 01 07 01 BC 01 0C 01 07 01 DC ....
0001C868 02 0C 01 07 01 44 0C 01 07 01 74 0C 01 07 01 80 .....D....t.....
0001C878 02 0C 01 07 01 9C 03 0C 01 07 01 A8 01 0C 01 07 ....
0001C888 01 A4 02 0C 01 07 01 A4 0A 0C 01 07 01 D0 02 0C ....
0001C898 01 07 01 C4 09 0C 01 07 01 88 09 0C 01 07 01 B8 ....
0001C8A8 0A 0C 01 07 01 D4 09 0C 01 07 01 F0 08 0C 01 07 ....
0001C8B8 01 98 0A 0C 01 07 01 8C 02 0C 01 07 01 B4 0A 0C ....
0001C8C8 01 07 01 D0 09 0C 01 07 01 FC 08 0C 01 07 01 94 ....
0001C8D8 0A 0C 01 07 01 C0 09 0C 01 07 01 94 09 0C 01 07 ....
0001C8E8 01 DC 01 0C 01 07 01 A0 0A 0C 01 07 01 B0 0A 0C ....
0001C8F8 01 07 01 C0 02 0C 01 07 01 B4 02 0C 01 07 01 84 ....
0001C908 0A 0C 01 07 01 8C 0A 0C 01 07 01 B8 09 0C 01 07 ....
0001C918 01 90 09 0C 01 07 01 A8 0A 0C 01 07 01 EC 0A 0C ....
0001C928 01 07 01 88 0A 0C 01 07 01 C8 0A 0C 01 07 01 B4 ....
0001C938 09 0C 01 07 01 E4 09 0C 01 07 01 A0 09 0C 01 07 ....
0001C948 01 F8 08 0C 01 07 01 F0 01 0C 01 07 01 AC 03 0C ....
0001C958 01 07 01 90 0A 0C 01 07 01 BC 09 0C 01 07 01 8C ....
0001C968 09 0C 01 07 01 9C 0A 0C 01 07 01 C8 09 0C 01 07 ....
0001C978 01 84 09 0C 01 07 0E B4 05 20 20 20 20 20 20 20 ....
0001C988 20 20 20 20 20 24 0C 01 07 01 A8 02 0C 01 07 02 $.........
0001C998 9C 04 38 0C 01 07 02 A8 04 38 0C 01 07 01 8C 03 ..8.....8.....
0001C9A8 0C 01 07 01 AC 0A 0C 01 07 01 CC 09 0C 01 07 01 ....
0001C9B8 80 09 0C 01 07 01 CC 03 0C 01 07 01 C0 03 0C 01 ....
0001C9C8 07 01 E8 03 0C 01 07 04 F0 03 08 0C 3C 01 05 04 ....<...
0001C9D8 00 02 05 03 84 03 88 01 38 06 03 88 03 88 01 38 ....8.....8
0001C9E8 14 02 05 01 A0 03 06 01 A4 03 0E 02 05 01 B4 03 ....
0001C9F8 06 01 B8 03 1B 02 05 02 94 04 38 06 02 98 04 38 ....8....8
0001CA08 01 06 01 00 02 05 02 8C 0B 34 06 02 90 0B 34 01 ....4....4.
0001CA18 08 02 98 03 02 05 01 EC 02 06 01 F0 02 08 02 05 ....
0001CA28 01 F8 02 06 01 FC 02 01 09 03 00 02 05 01 30 06 ....0.
0001CA38 01 34 80 01 02 05 01 94 01 06 01 98 01 20 02 05 .4.......... ..
0001CA48 01 C8 01 06 01 CC 01 02 08 3F 00 02 05 01 00 06 ....?......
0001CA58 01 04 08 02 05 01 0C 06 01 10 08 02 05 01 18 06 ....
0001CA68 01 1C 08 02 05 01 24 06 01 28 08 02 05 01 30 06 ......$..(....0.
0001CA78 01 34 08 02 05 01 3C 06 01 40 08 02 05 01 48 06 .4....<..@....H.
0001CA88 01 4C 08 02 05 01 54 06 01 58 08 02 05 01 60 06 .L....T..X....`.
0001CA98 01 64 08 02 05 01 6C 06 01 70 08 02 05 01 78 06 .d....l..p....x.
0001CAA8 01 7C 08 02 05 01 84 01 06 01 88 01 08 02 05 01 .|..............
0001CAB8 90 01 06 01 94 01 08 02 05 01 9C 01 06 01 A0 01 ....
0001CAC8 08 02 05 01 A8 01 06 01 AC 01 08 02 05 01 B4 01 ....

Какими инструментами будем пользоваться?

Xcode

Apple Opensource (XNU)

Различные Mach-O анализаторы

```
/*
 * The 64-bit segment load command indicates that a part of this file is to be
 * mapped into a 64-bit task's address space.  If the 64-bit segment has
 * sections then section_64 structures directly follow the 64-bit segment
 * command and their size is reflected in cmdsize.
 */
struct segment_command_64 {     /* for 64-bit architectures */
    uint32_t    cmd;            /* LC_SEGMENT_64 */
    uint32_t    cmdsize;        /* includes sizeof section_64 structs */
    char        segname[16];    /* segment name */
    uint64_t    vmaddr;         /* memory address of this segment */
    uint64_t    vmsize;         /* memory size of this segment */
    uint64_t    fileoff;        /* file offset of this segment */
    uint64_t    filesize;       /* amount to map from the file */
    vm_prot_t   maxprot;        /* maximum VM protection */
    vm_prot_t   initprot;       /* initial VM protection */
    uint32_t    nsects;         /* number of sections in segment */
    uint32_t    flags;          /* flags */
};
```

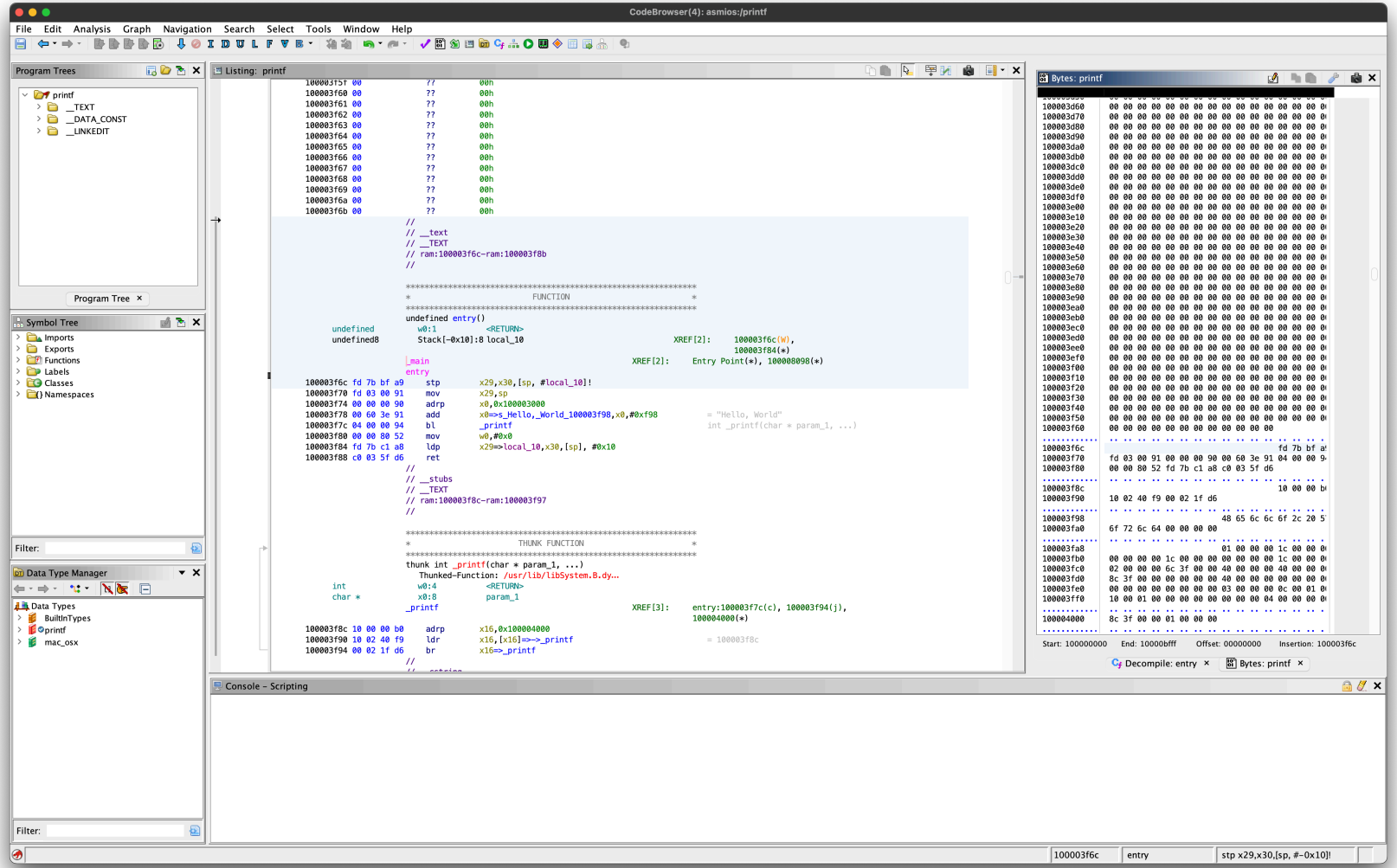
Какими инструментами будем пользоваться?

Xcode

Apple Opensource (XNU)

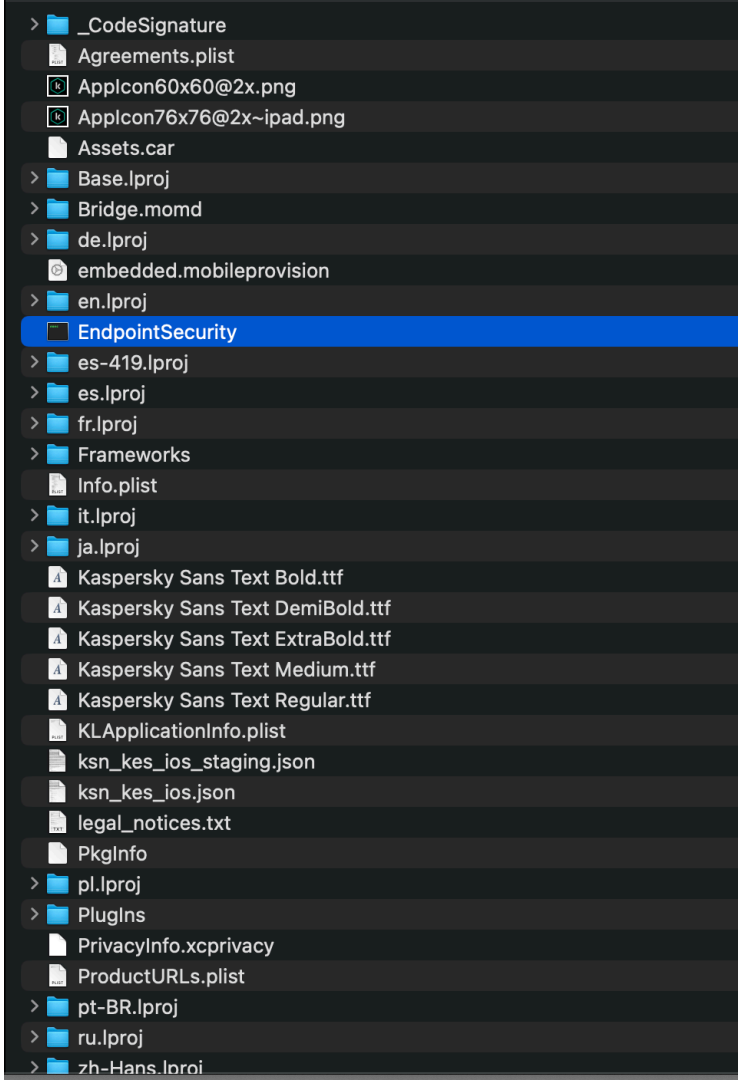
Различные Mach-O анализаторы

Ghidra


CodeBrowser(4): asmios:/printf
File Edit Analysis Graph Navigation Search Select Tools Window Help
Program Trees
printf
__TEXT
__DATA_CONST
__LINKEDIT
Program Tree
Symbol Tree
Imports
Exports
Functions
Labels
Classes
Namespaces
Filter:
Data Type Manager
Data Types
BuiltInTypes
printf
mac_osx
Filter:
Listing: printf
100003f5f 00 ?? 00h
100003f60 00 ?? 00h
100003f61 00 ?? 00h
100003f62 00 ?? 00h
100003f63 00 ?? 00h
100003f64 00 ?? 00h
100003f65 00 ?? 00h
100003f66 00 ?? 00h
100003f67 00 ?? 00h
100003f68 00 ?? 00h
100003f69 00 ?? 00h
100003f6a 00 ?? 00h
100003f6b 00 ?? 00h
//
// __text
// __TEXT
// ram:100003f6c-ram:100003f8b
//
FUNCTION
undefined entry()
undefined w0:1 <RETURN>
undefined8 Stack[-0x10]:8 local_10 XREF[2]: 100003f6c(W), 100003f84(*)
_main XREF[2]: Entry Point(*), 100008098(*)
entry
100003f6c fd 7b bf a9 stp x29,x30,[sp, #local_10]!
100003f70 fd 03 00 91 mov x29,sp
100003f74 00 00 00 90 adrp x0,0x100003000
100003f78 00 60 3e 91 add x0=>s_Hello,_World_100003f98,x0,#0xf98 = "Hello, World"
100003f7c 04 00 00 94 bl _printf int _printf(char * param_1, ...)
100003f80 00 00 80 52 mov w0,#0x0
100003f84 fd 7b c1 a8 ldp x29=>local_10,x30,[sp], #0x10
100003f88 c0 03 5f d6 ret
//
// __stubs
// __TEXT
// ram:100003f8c-ram:100003f97
//
THUNK FUNCTION
thunk int _printf(char * param_1, ...)
Thunked-Function: /usr/lib/libSystem.B.dy...
int w0:4 <RETURN>
char * x0:8 param_1
_printf XREF[3]: entry:100003f7c(c), 100003f94(j), 100004000(*)
100003f8c 10 00 00 b0 adrp x16,0x100004000
100003f90 10 02 40 f9 ldr x16,[x16]=>_printf = 100003f8c
100003f94 00 02 1f d6 br x16=>_printf
//
Bytes: printf
100003d60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003d70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003d80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003d90 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003da0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003db0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003dc0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003dd0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003de0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003df0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003e00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003e10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003e20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003e30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003e40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003e50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003e60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003e70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003e80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003e90 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003ea0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003eb0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003ec0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003ed0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003ee0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003ef0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003f00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003f10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003f20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003f30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003f40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003f50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100003f60 00 00 00 00 00 00 00 00 00 00 00 00
100003f6c fd 7b bf a9
100003f70 fd 03 00 91 00 00 00 90 00 60 3e 91 04 00 00 94
100003f80 00 00 80 52 fd 7b c1 a8 c0 03 5f d6
100003f8c 10 00 00 b0
100003f90 10 02 40 f9 00 02 1f d6
100003f98 48 65 6c 6c 6f 2c 20 57
100003fa0 6f 72 6c 64 00 00 00 00
100003fa8 01 00 00 00 1c 00 00 00
100003fb0 00 00 00 00 1c 00 00 00 00 00 00 00 1c 00 00 00
100003fc0 02 00 00 00 6c 3f 00 00 40 00 00 00 40 00 00 00
100003fd0 8c 3f 00 00 00 00 00 00 40 00 00 00 00 00 00 00
100003fe0 00 00 00 00 00 00 00 00 03 00 00 00 0c 00 01 00
100003ff0 10 00 01 00 00 00 00 00 00 00 00 00 04 00 00 00
100004000 8c 3f 00 00 01 00 00 00
Start: 100000000 End: 10000bfff Offset: 00000000 Insertion: 100003f6c
Decompile: entry
Bytes: printf
Console - Scripting
100003f6c entry stp x29,x30,[sp, #-0x10]!

iOS App

Что внутри iOS приложения?


_CodeSignature
Agreements.plist
AppIcon60x60@2x.png
AppIcon76x76@2x~ipad.png
Assets.car
Base.lproj
Bridge.momd
de.lproj
embedded.mobileprovision
en.lproj
EndpointSecurity
es-419.lproj
es.lproj
fr.lproj
Frameworks
Info.plist
it.lproj
ja.lproj
Kaspersky Sans Text Bold.ttf
Kaspersky Sans Text DemiBold.ttf
Kaspersky Sans Text ExtraBold.ttf
Kaspersky Sans Text Medium.ttf
Kaspersky Sans Text Regular.ttf
KLApplicationInfo.plist
ksn_kes_ios_staging.json
ksn_kes_ios.json
legal_notices.txt
PkgInfo
pl.lproj
PlugIns
PrivacyInfo.xcprivacy
ProductURLs.plist
pt-BR.lproj
ru.lproj
zh-Hans.lproj

Что внутри iOS приложения?

Ресурсы (картинки, строки, шрифты, …)

Фреймворки и Extension-ы

Информация о подписи

Исполняемый бинарный файл

Как запускается .app?

Запускает приложение ОС

Загрузить основной бинарь

Проверить, можно ли вообще его запускать

Загрузить необходимые для старта либы

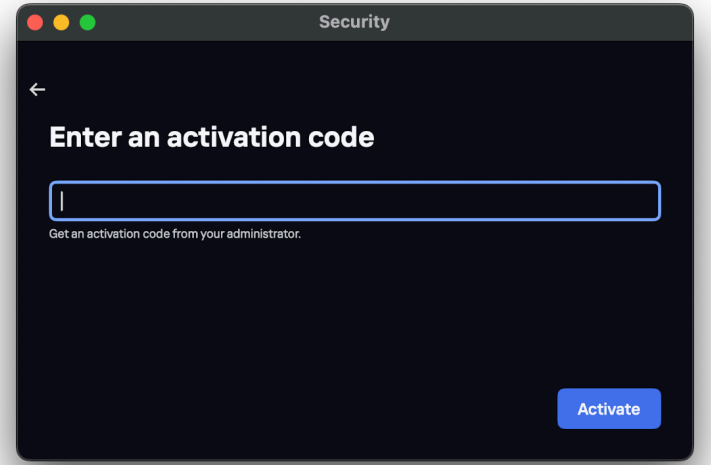
Как запускается .app?

Порезолвить все внешние (а иногда и внутренние) символы

Начать выполнение бинаря с Entrypoint (как правило _main)

Почему нельзя взять и запустить iOS приложение на маке?

Во-первых, можно


Security
Enter an activation code
Get an activation code from your administrator.
Activate

Почему нельзя взять и запустить iOS приложение на маке?

Во-первых, можно

Можно запустить arm64 ipa на M* процессорах

Но можно запустить только если совпадают архитектуры

Достаточно добавить одну галочку в настройках

А что с intel?

А вот на intel уже такой фокус не получится

ОС это видит, и не даёт запустить бинарь

Архитектура бинарника в ipa arm64, а intel x86

Инструкции в бинарнике от другой архитектуры

Что будем делать мы?

Intel-ом мы заниматься не будем, тк нужно будет транслировать все инструкции

Попробуем воспроизвести запуск arm64 ipa на M*

Mach-O

Пару слов про Mach-O

Просто бинарный формат файла

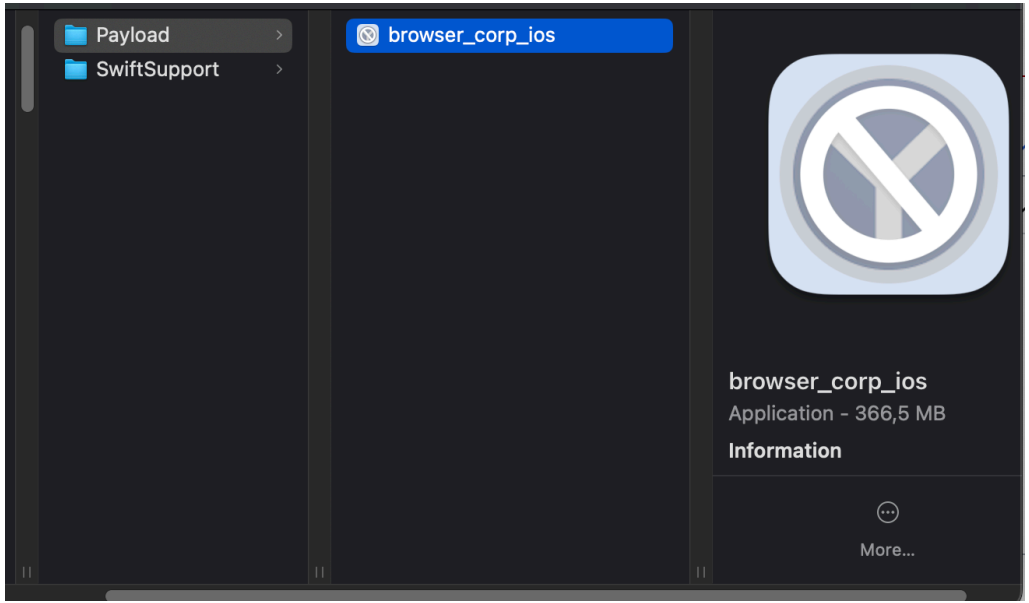
Содержит библиотеки

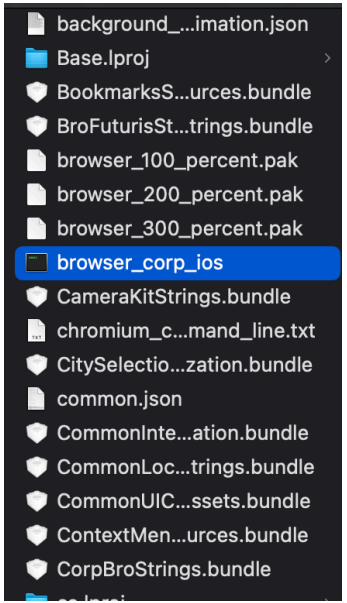
Содержит исполняемые файлы

Это тот бинарник, который вы видите внутри app

Пару слов про Mach-O

Указывает как загрузить и
запустить бинарник


Payload
SwiftSupport
browser_corp_ios
browser_corp_ios
Application - 366,5 MB
Information
More...


background_...imation.json
Base.lproj
BookmarksS...urces.bundle
BroFuturisSt...trings.bundle
browser_100_percent.pak
browser_200_percent.pak
browser_300_percent.pak
browser_corp_ios
CameraKitStrings.bundle
chromium_c...mand_line.txt
CitySelectio...zation.bundle
common.json
CommonInte...ation.bundle
CommonLoc...trings.bundle
CommonUIC...ssets.bundle
ContextMen...urces.bundle
CorpBroStrings.bundle

Где используется Mach-O?

Только в OS от Apple

macOS

iOS

…

Супер коротко о формате

Команды для загрузчика

Информация о символах

“Сырые” байты бинарника с инструкциями

Мета-информация (например, для какой OS бинарник, уникальный id)

Binary Loader

dyld

DYLD(1)

NAME

dyld - the dynamic linker

Что должен делать загрузчик

Разобрать Mach-O формат

Подгрузить зависимости

Выполнить команды (например, по загрузке таблицы символов)

Загрузить исполняемый файл в память

Что должен делать загрузчик

Порезолвить все undefined символы

Переместить указатель на Entrypoint

Пометить память как R/X

Начать выполнение исполняемого файла

“Идеальный” код

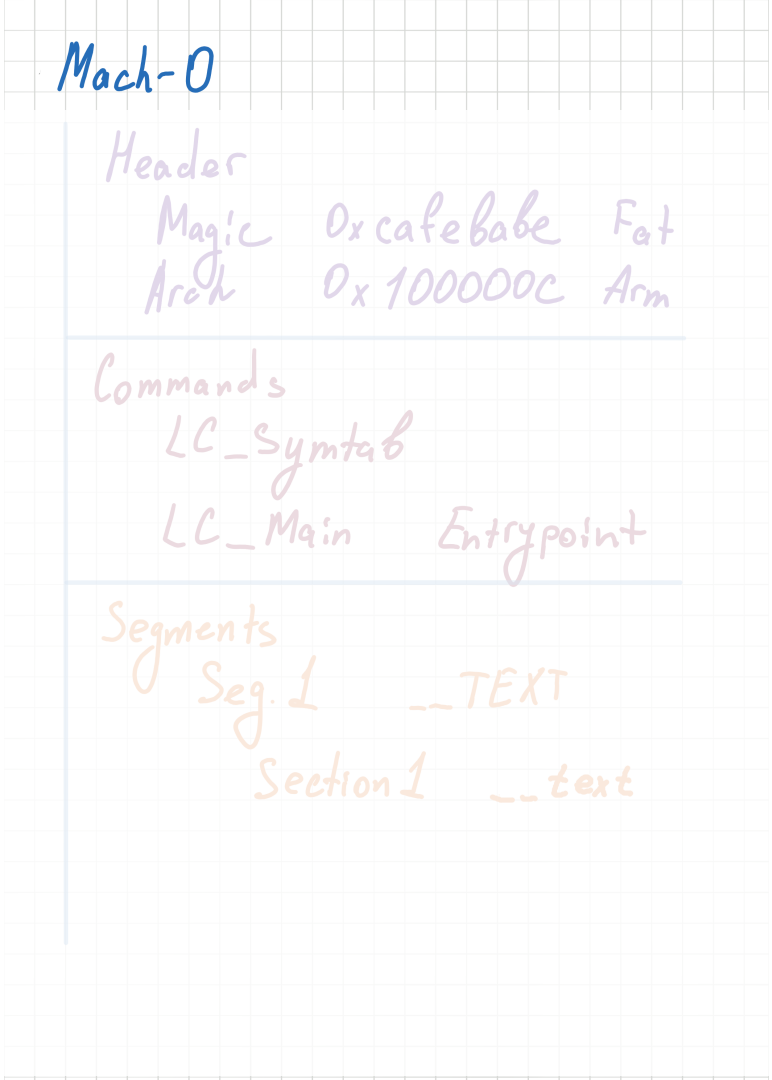
```
func execute() {
    // Parse
    let analyzer = …
    let analyzedMachO = …

    // Load
    let loader = …
    let entrypoint = …

    // Execute
    let vm = VM()
    vm.jmp(entrypoint.address)
}
```

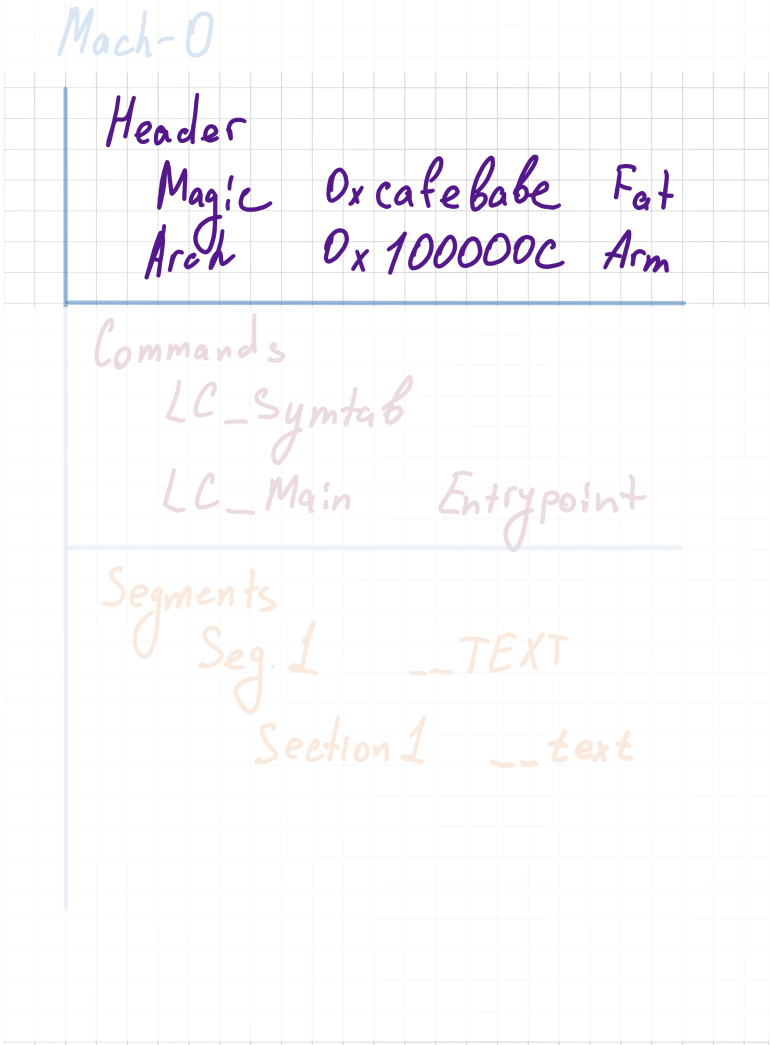
Реальный код

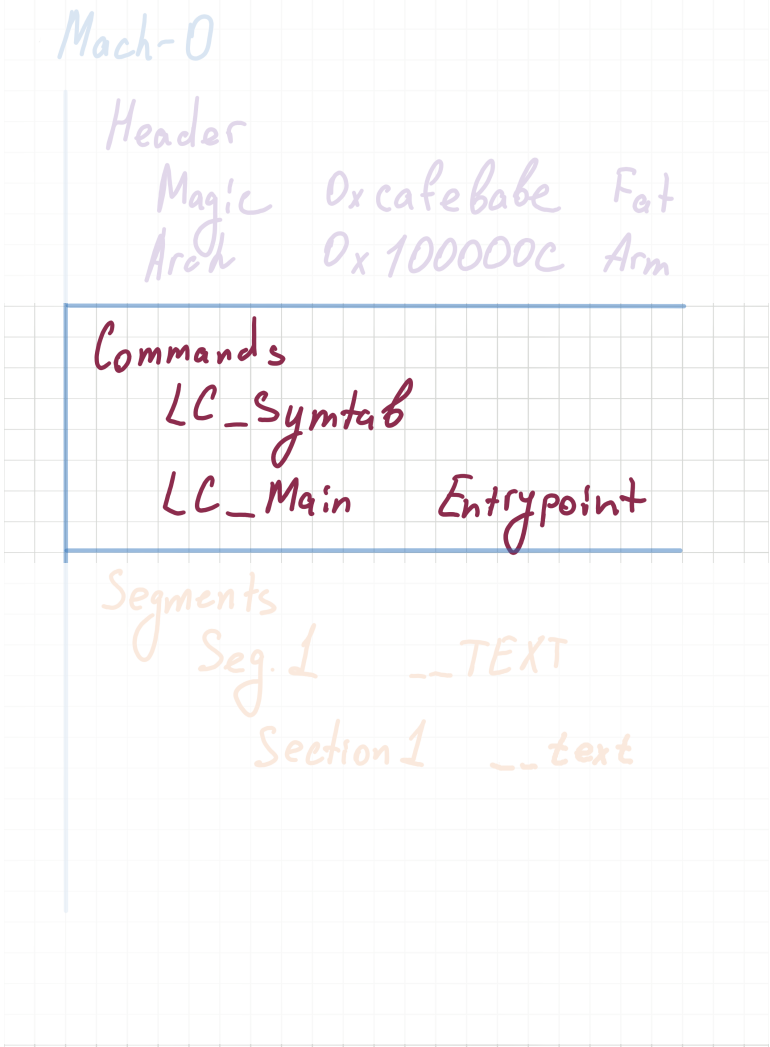
```
func execute() throws {
    executorActivity.info("Open \(fileUrl.path)")
    // Parse
    let analyzer = try SimpleBinaryAnalyzer(parentActivity: executorActivity, fileURL: fileUrl)
    let analyzedMachO = try analyzer.analyze().first(where: { $0.binary.isArm64 })!

    // Load
    let functionsHandler = FunctionsHandler()
    let functionsTable = functionsHandler.functionsTable()
         .reduce(into: [String: Int]()) { $0[$1.key] = $1.value.intValue }
    let loadedLibrariesStorage = LoadedLibrariesStorage()
    let loader = MachOLoader.Loader(parentActivity: executorActivity,
                                    externaFunctionImplementations: functionsTable,
                                    objectFile: analyzedMachO.binary,
                                    objectFileReader: analyzedMachO.reader,
                                    loadedLibrariesStorage: loadedLibrariesStorage)


    let loadedBinary = try loader.load()
    if let executable = loadedBinary as? LoadedExecutable {
        // Execute
        let vm = VM()
        vm.jmp(executable.entrypoint.address)
    }
    else if let library = loadedBinary as? LoadedLibrary {
        executorActivity.info("Loaded library: \(library)")
    }
    else {
        executorActivity.failure("Unsupported binary type: \(analyzedMachO.binary.header.fileType)")
    }
}
```

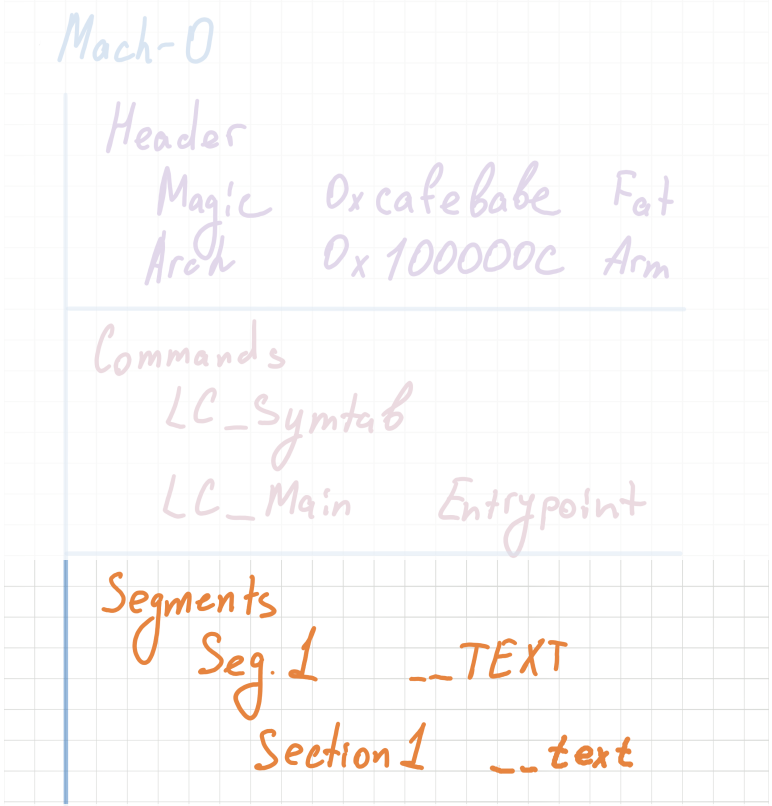
Строение Mach-O

Mach-O


Header
Magic 0xcafebabe Fat
Arch 0x100000c Arm
Commands
LC_Symtab
LC_Main Entrypoint
Segments
Seg. 1 __TEXT
Section 1 __text


Mach-O
Header
Magic 0xcafebabe Fat
Arch 0x100000c Arm
Commands
LC_Symtab
LC_Main Entrypoint
Segments
Seg. 1 __TEXT
Section 1 __text


Mach-O
Header
Magic 0xcafebabe Fat
Arch 0x100000c Arm
Commands
LC_Symtab
LC_Main Entrypoint
Segments
Seg. 1 __TEXT
Section 1 __text


Mach-O
Header
Magic 0xcafebabe Fat
Arch 0x100000c Arm
Commands
LC_Symtab
LC_Main Entrypoint
Segments
Seg. 1 __TEXT
Section 1 __text

Команды (основных около 10)

Получение мета-информации

LC_UUID, LC_RPATH,
LC_ID_DYLIB, …

Загрузка таблицы символов

LC_SYMTAB

Загрузка бинарных данных

LC_SEGMENT /
LC_SEGMENT_64

Symbols

Name	Type	Value
Symbol 1	Undefined	0x0
Symbol 2	Public External	0x12345
Symbol 3	Private External	0x54321
Symbol 4	Stab	0xABCDE

Symbols

Name	Type	Value
Symbol 1	Undefined	0x0
Symbol 2	Public External	0x12345
Symbol 3	Private External	0x54321
Symbol 4	Stab	0xABCDE

Symbols

Name	Type	Value
Symbol 1	Undefined	0x0
Symbol 2	Public External	0x12345
Symbol 3	Private External	0x54321
Symbol 4	Stab	0xABCDE

Symbols

Name	Type	Value
Symbol 1	Undefined	0x0
Symbol 2	Public External	0x12345
Symbol 3	Private External	0x54321
Symbol 4	Stab	0xABCDE

Symbols

Name	Type	Value
Symbol 1	Undefined	0x0
Symbol 2	Public External	0x12345
Symbol 3	Private External	0x54321
Symbol 4	Stab	0xABCDE

Symbols

Name	Type	Value
Symbol 1	Undefined	0x0
Symbol 2	Public External	0x12345
Symbol 3	Private External	0x54321
Symbol 4	Stab	0xABCDE

Команды (всего основных около 10)

Получение мета информации

LC_UUID, LC_RPATH, LC_ID_DYLIB, …

Загрузка таблицы символов

LC_SYMTAB

Загрузка бинарных данных

LC_SEGMENT / LC_SEGMENT_64

Команда на резолв внешних символов

LC_DYLD_CHAINED_FIXUPS

Memory

Приходите к нам в Swift, у нас есть:

UnsafePointer

UnsafeMutableRawPointer

UnsafeMutableRawBufferPointer

UnsafeRawPointer

UnsafeBufferPointer

<τ_0_0>()

UnsafeMutablePointer

UnsafeMutableBufferPointer

Что будем делать с памятью

“Переложим” байты из файла в RAM по смещениям

Переключим режим доступа к памяти (rwx)

Поговорим про ASLR

ASLR

Address Space Layout Randomization

Изменяется адрес стека, кучи, и динамических библиотек

Каждый запуск — разные участки памяти

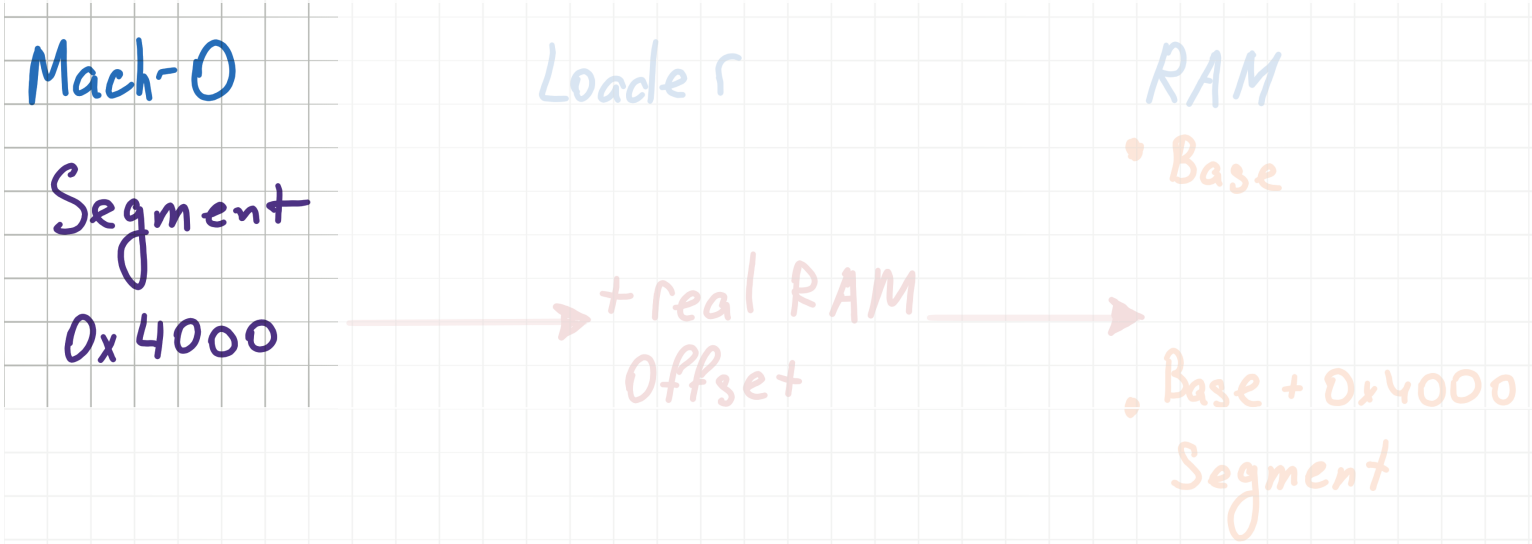
Помогает бороться с overflow уязвимостями

Влияние ASLR

ASLR

Mach-O
Loader
RAM
Segment
0x4000
+ real RAM
Offset
Base
Base + 0x4000
Segment
Segment 0x4000
RAM Start 0x10000
Segment 0x14000

ASLR

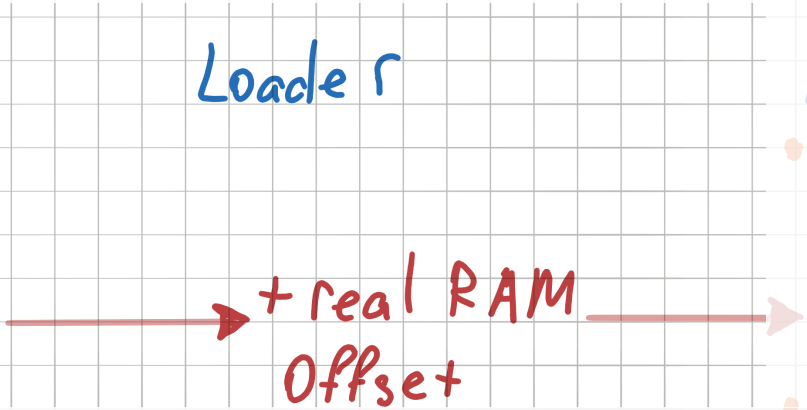

Mach-O
Segment
0x4000
Loader
+ real RAM
Offset
RAM
Base
Base + 0x4000
Segment
Segment 0x4000
RAM Start 0x10000
Segment 0x14000

ASLR

Mach-O

Segment

0x4000



RAM

- Base
- Base + 0x4000

Segment

Segment 0x4000

RAM Start 0x10000

} Segment 0x14000

ASLR

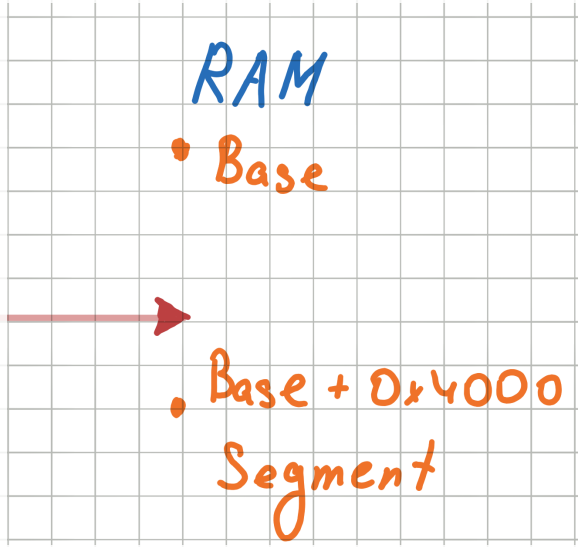
Mach-O

Loader

Segment

0x4000

+ real RAM offset



Segment 0x4000

RAM Start 0x10000

Segment 0x14000

ASLR

Mach-O

Segment

0x4000

Loader

+ real RAM

Offset

RAM

- Base
- Base + 0x4000

Segment

Segment 0x4000

RAM Start 0x10000

} Segment 0x14000

Влияние ASLR

Держим в уме, что все адреса будут сдвинуты на некоторый Base Address

Base Address

Откуда мы его возьмём?

Возьмём значение указателя
на старт буфера

Просто инициализируем
память для процесса

Будем использовать этот старт
в качестве Base Offset

```
let virtualMemoryBuffer = UnsafeMutableRawBufferPointer
  .allocate(byteCount: alignedSize,
            alignment: pageSize)
```

```
memset(virtualMemoryBuffer.baseAddress!, 0x0, Int(alignedSize))
```

Права доступа на память

Нельзя так просто взять и начать исполнять инструкции из RAM

Нужно предварительно на эту память выдать разрешения

В этом поможет memprotect

Права доступа на память

```
let protection: Int32 = PROT_READ | PROT_EXEC

mprotect(virtualMemoryBuffer.baseAddress,
         Int(virtualMemory.size),
         protection)
```

Как начать выполнять код?

```
- (int) jmp:(NSInteger)address
{
    int (*jump_function)(void) = (int (*)(void))address;
    int result = jump_function();
    return result;
}
```

Команды

LC_SEGMENT(64)

Команда загрузки сегмента в память

Статические данные

Реальный код вашей программы

Все сегменты поделены на секции

LC_SYMTAB

Таблица символов

А так же адрес, где искать реализацию (правильное слово — значение)

В ней содержится оффсет на имя символа

LC_MAIN

Содержит самое важное — оффсет на Entrypoint

Нужна по сути только для Executable типов бинарников

А так же размер стека

Давайте уже что-то запускать

Давайте, а что?

Для начала — приложение macOS на macOS :)

Можно его выполнить просто

./binary_name

А мы загрузим и выполним сами

Простое приложение для macOS

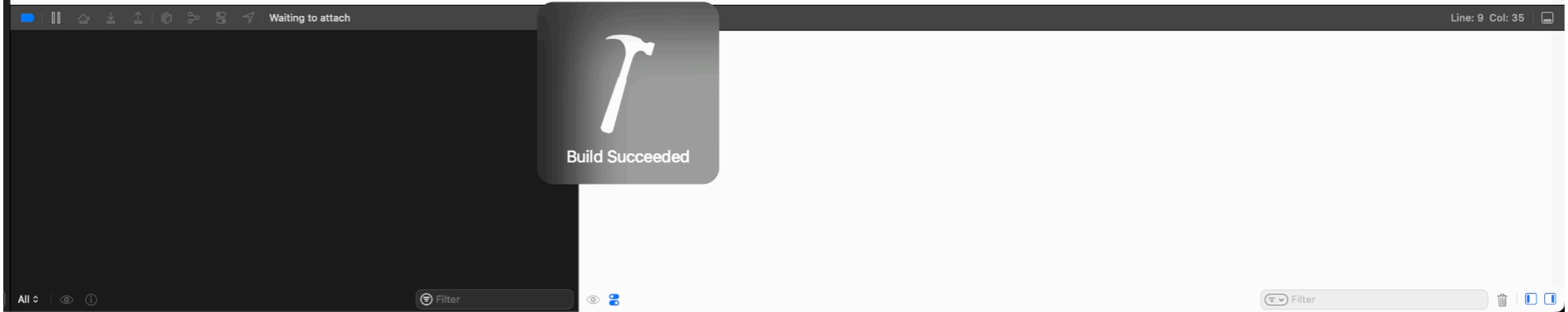
Как выглядит самое простое приложение?

```c
int main()
{
    return 123;
}
```

```
gcc ./main.c -o main
```

Давайте пробовать запускать

```
10      int result = jump_function();
11      NSLog(@"result: %d", result);
12  }
13
14  @end
15
```



Что-то посложнее?

```c
#include <stdio.h>

void print_please()
{
    printf("Hello, World!\n");
}

int main() {
    print_please();
}
```

```
gcc ./printf.c -o printf
```

```
                "Loading section: \(section.sectname). Offset in file: \(section.offset.hexRepresentation) Size: \(section.size)"
            )
            try objectFileReader.setVirtualOffset(section.offset)
            let sectionContent = try objectFileReader.readNext(section.size)

            let sectionAddressInRam = executable ? section.addr - 0x100000000 : section.addr
            let sectionEditor = try segmentEditor.baseReader().child(startingAt: sectionAddressInRam - segmentEditor.realOffset, size: section.size)

            let hexRepresentation = try sectionEditor.hexRepresentation()
            loaderActivity.verbose("\(hexRepresentation)")
        }
    }

    let hexR
    loaderAc

    let useF
    if useFi
//          le
//          tr                                                    : virtualMemoryBuffer.baseAddress!))
    }
    else {
        // T
        try

    }

    let protection: Int32 = PROT_READ | PROT_EXEC
```

CustomSimulator

Select file.

Open

Will never be executed

CustomSimulator

Line: 116 Col: 25

<V> [Root] Started

Filter

Filter

Что-то пошло не так…

Чем отличаются два наших бинаря?

Одному из них нужна функция printf, которую нужно откуда-то подгрузить

Dynamic linking

Как подгрузить динамическую библиотеку?

Нужно знать её имя

Загрузить её в RAM

Где она находится

Получить адреса функций, которые в ней определены

Что не хватает нашему бинарю?

Есть команда LC_LOAD_DYLIB (libSystem.B.dylib)

Подставить адрес этого символа в наш бинарь

Символ _printf (остальные не интересуют)

Symbol Resolution

Binary | Loader | RAM

_printf
Undefined

Знаем Location

1. Load libsystem.dylib
2. Get Addr (_printf)
3. Replace Location

_printf
(из libsystem)

Symbol Resolution

Binary

_printf
Undefined

Знаем Location

Loader

1. Load libsystem.dylib
2. Get Addr (_printf)
3. Replace Location

RAM

_printf
(из libsystem)

Symbol Resolution

Binary

_printf
Undefined

Знаем Location

Loader

1. Load libsystem.dylib
2. Get Addr (_printf)
3. Replace Location

RAM

_printf
(из libsystem)

Symbol Resolution

Binary

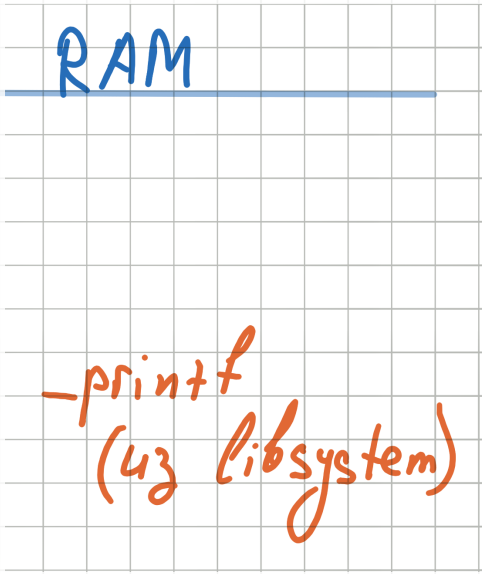
_printf
Undefined

Known Location

Loader

1. Load libsystem.dylib
2. Get Addr (_printf)
3. Replace Location

RAM



Let’s do it

```
for command in allLoadSegmentCommands.filter({ !$0.segname.contains("__PAGEZERO") }) {
    loaderActivity.verbose("Loading segment: \(command.segname)")
    let segmentStartAddress = executable ?  command.vmaddr - 0x100000000 : command.vmaddr
    let segmentEditor = try virtualMemory.baseReader().child(startingAt: segmentStartAddress, size: command.vmsize)
    for section in command.sections {
                                                                                    entation) Size: \(section.size)"
                                                                                    Ram - segmentEditor.realOffset, size: section.size)
    }
}

let hexR
loaderAc

let useFixupChains = true
if useFixupChains {
    let fixups = objectFile.getLcChainedFixupsCommands()
    try resolveFixups(fixups: fixups, wholeVm: virtualMemory, baseVmAddress: Int(bitPattern: virtualMemoryBuffer.baseAddress!))
```

CustomSimulator

Select file.

Open

CustomSimulator

Line: 114 Col: 67

```
<V>  [Root] Started
```

Filter

Filter

Какие особенности у библиотек

Существует Shared Cache

К ним обращаются App / Libs

Грузятся один раз

Подгружаются “особенным” способом

Own Library impl

Что ещё можно делать интересного?

Знаем имя вызываемой функции

Можем подменить реализацию на свою

Знаем адрес

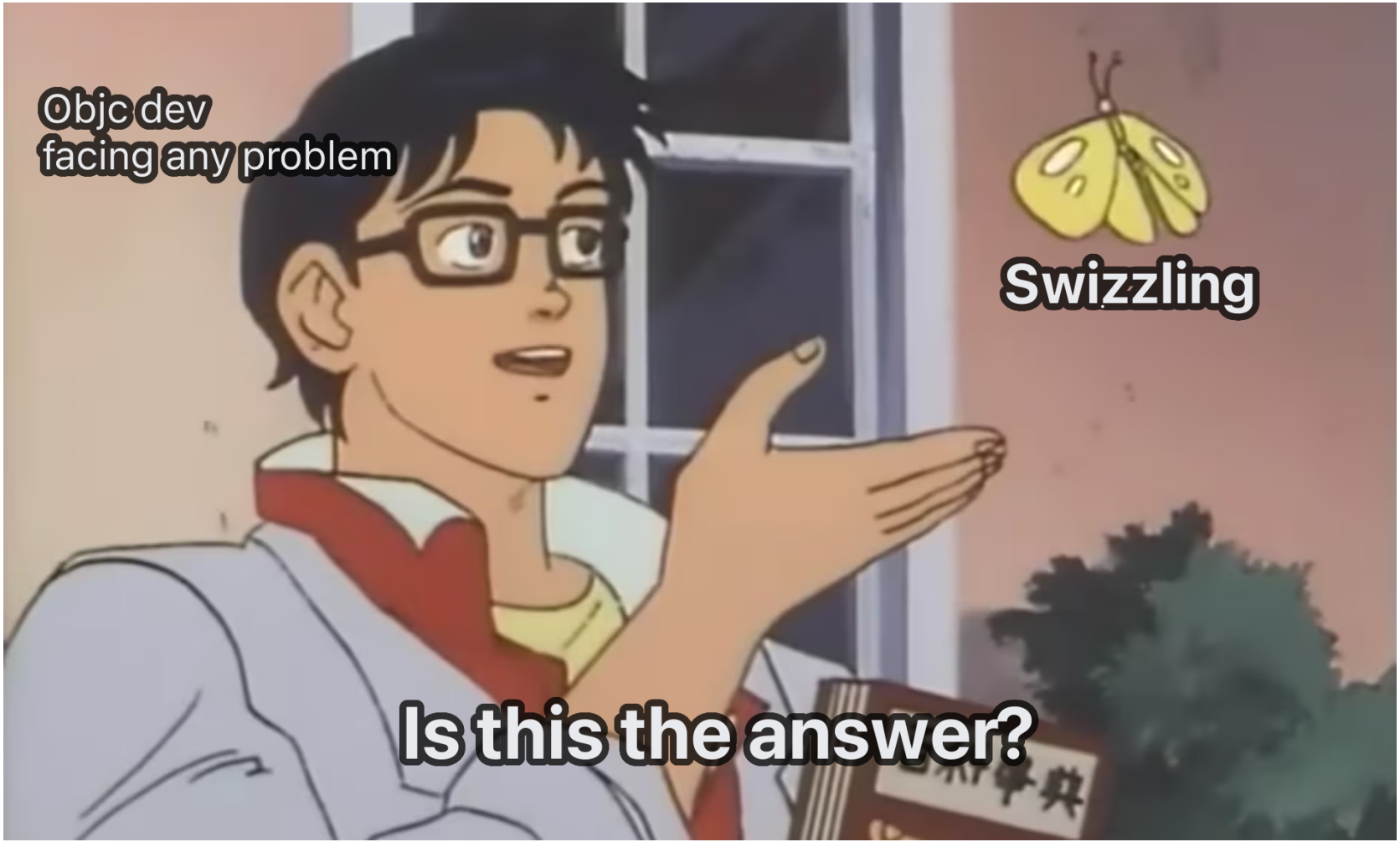
Ничего не напоминает?

Objc

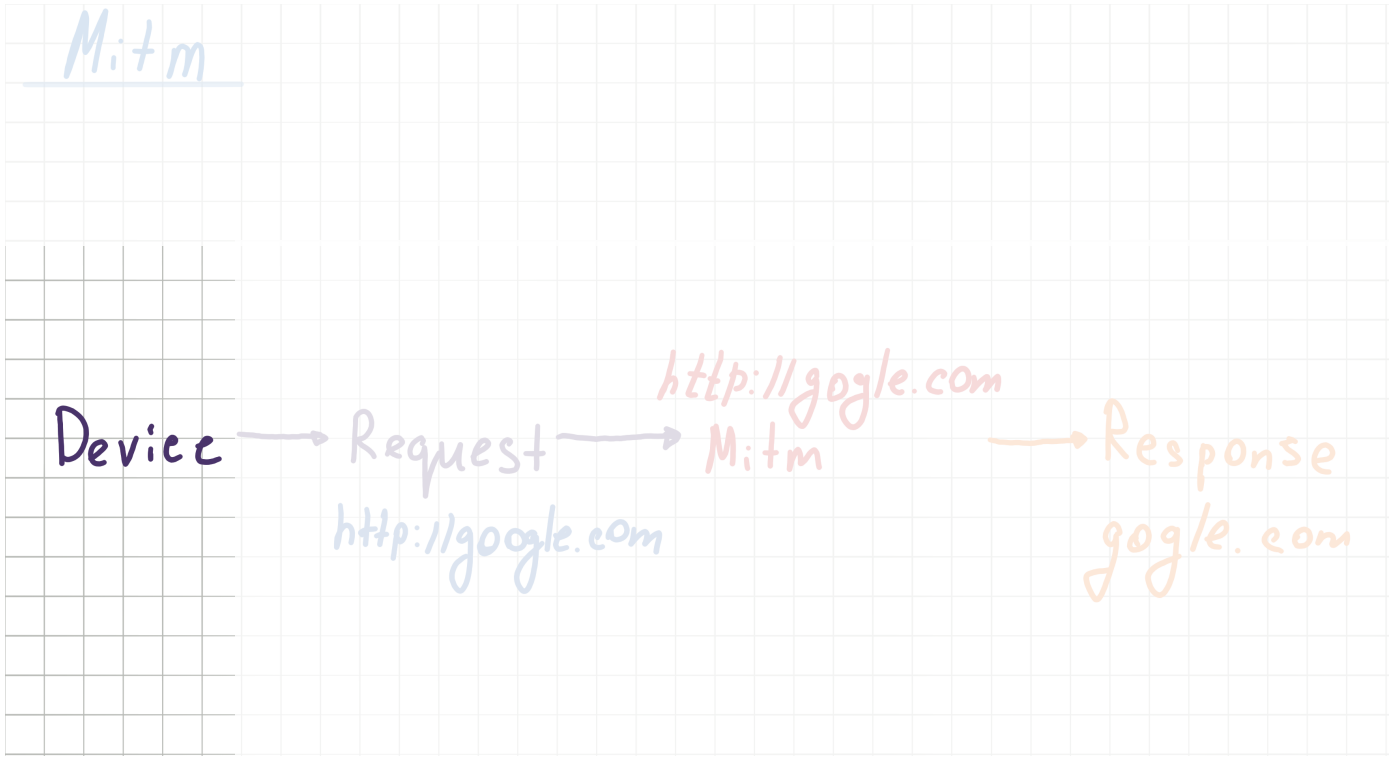
@selector

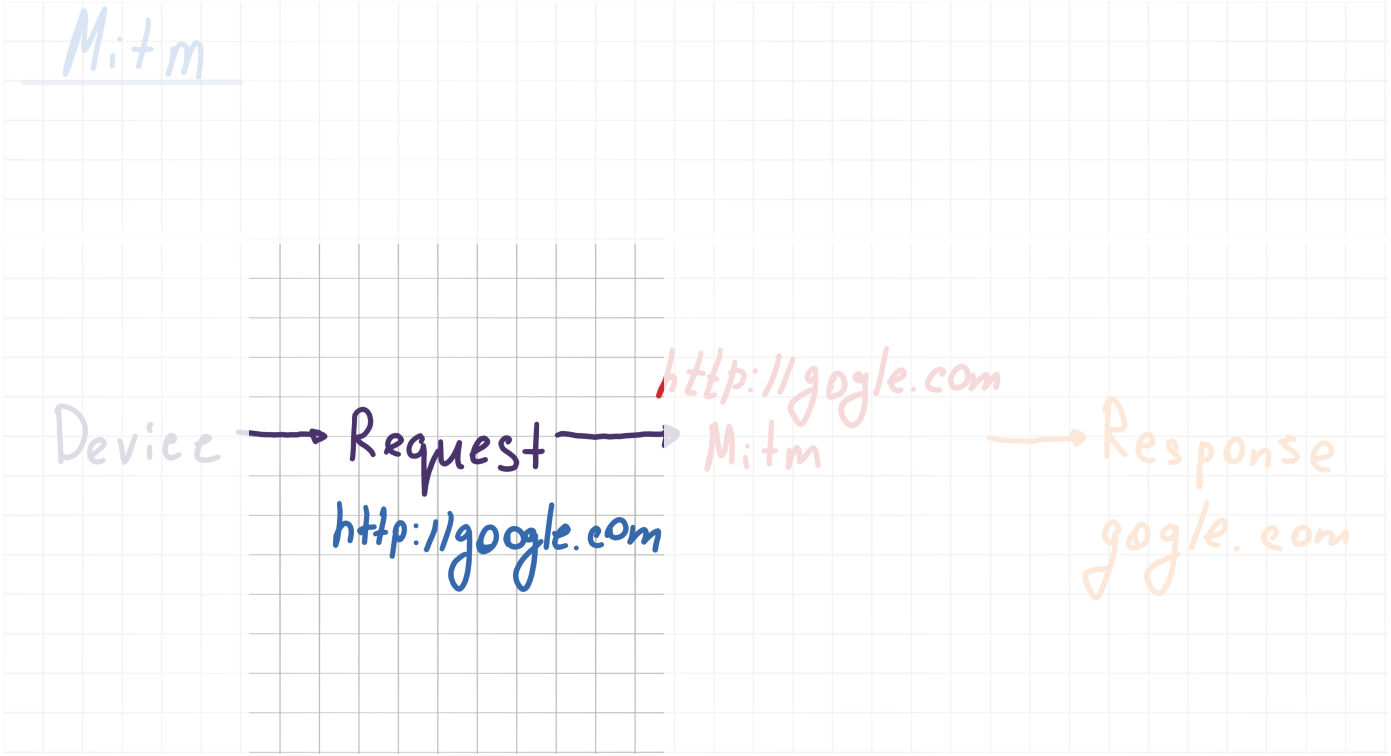
SEL

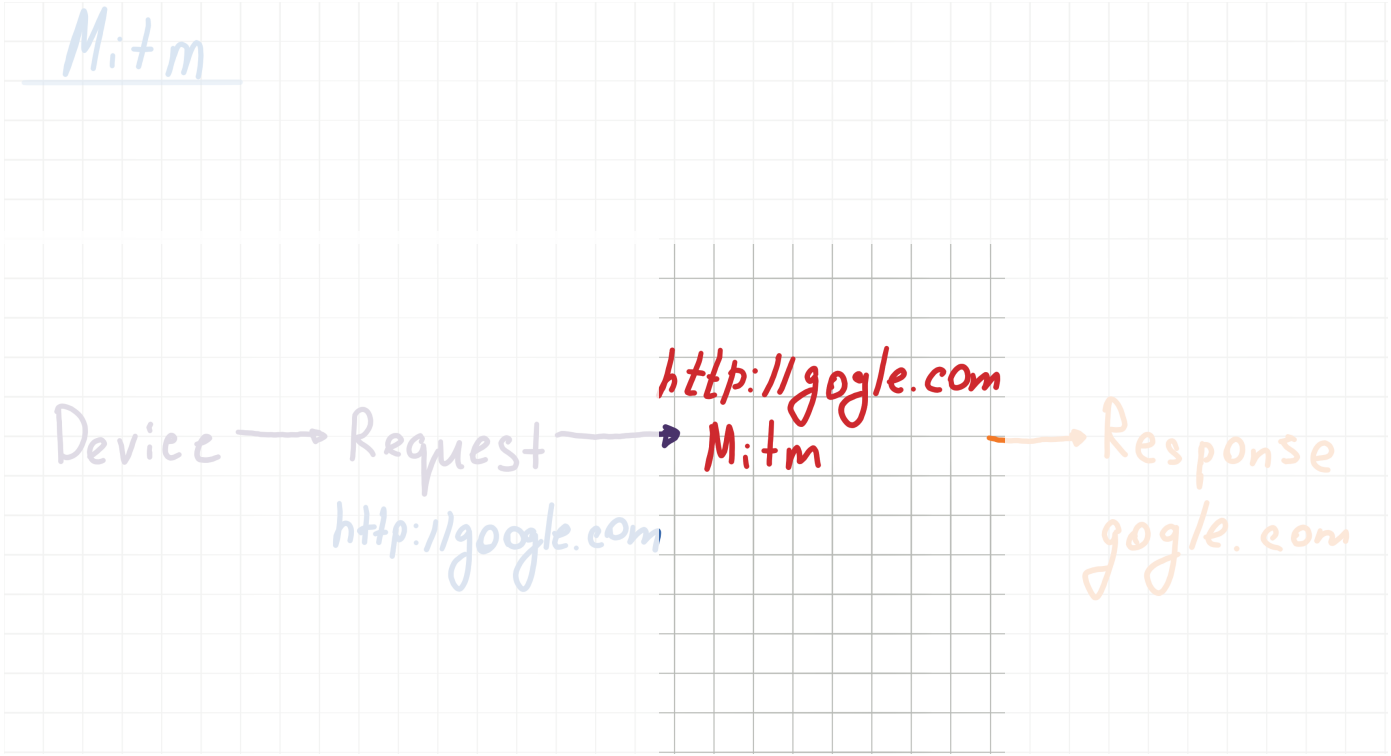
IMP

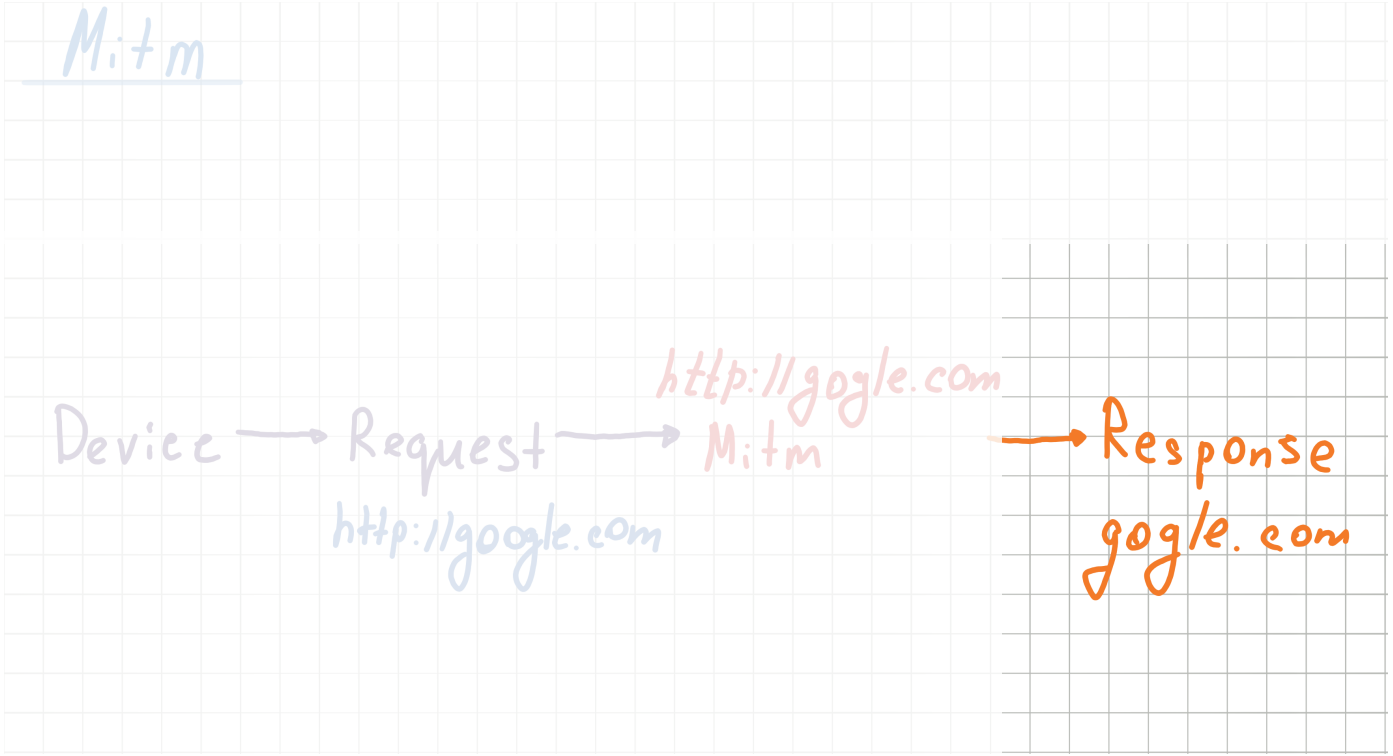

Objc dev
facing any problem
Swizzling
Is this the answer?

Mitm


Mitm
Device
Request
http://google.com
http://gogle.com
Mitm
Response
gogle.com


Mitm
Device
Request
http://google.com
http://gogle.com
Mitm
Response
gogle.com


Mitm
Device
Request
http://google.com
http://gogle.com
Mitm
Response
gogle.com

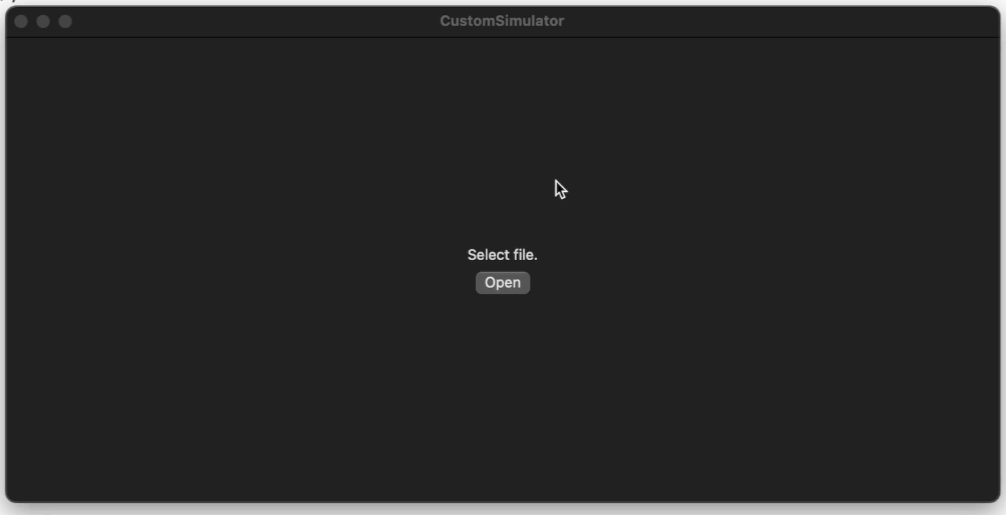

Mitm
Device
Request
http://google.com
http://gogle.com
Mitm
Response
gogle.com

Let’s do it [2]

```c
void function_printf(char *format,...)
{
    printf("MY PRINTF\n");

    va_list args;
    va_start(args, format);

    vprintf(format, args);

    va_end(args);
}
```

```
void *function_pointer = (void*)function_printf;
```

```
    va_start(args, format);

    vprintf(format, args);

    va_end(args);
}
```



CustomSimulator

Line: 10 Col: 28

```
<V>  [Root] Started
```

All

Filter

Filter

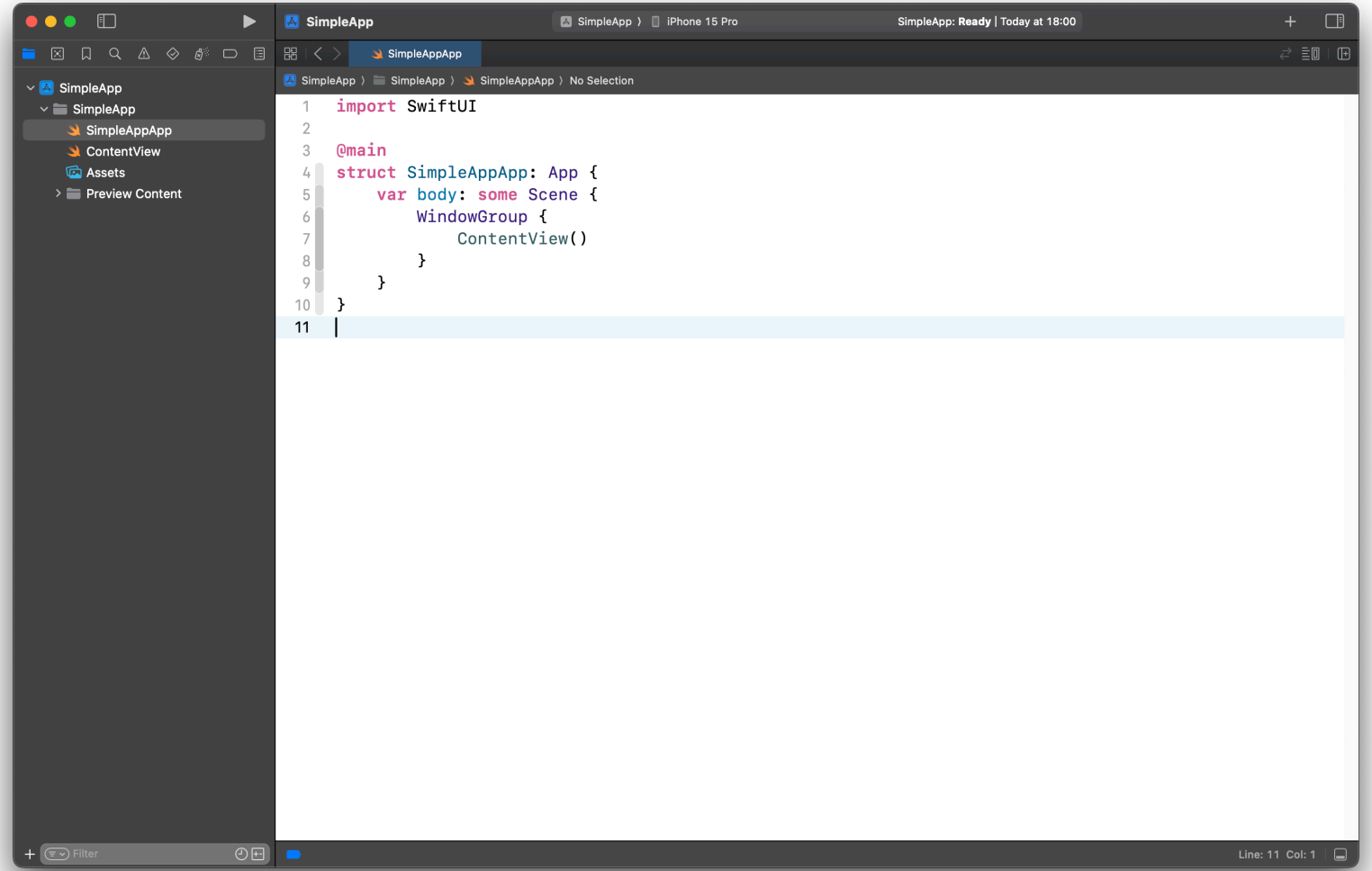
Результаты?

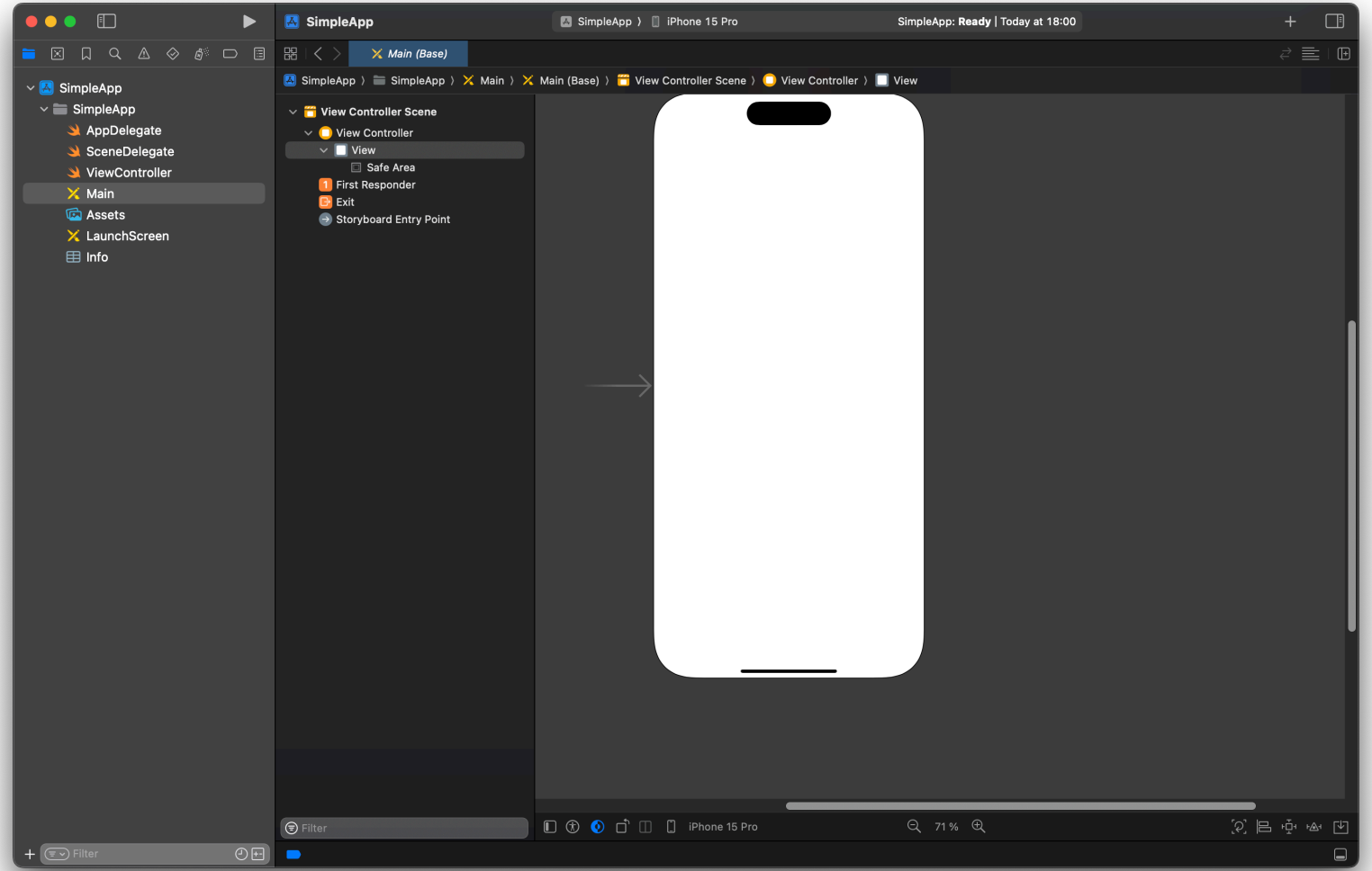
Мы научились запускать
macOS приложение на macOS

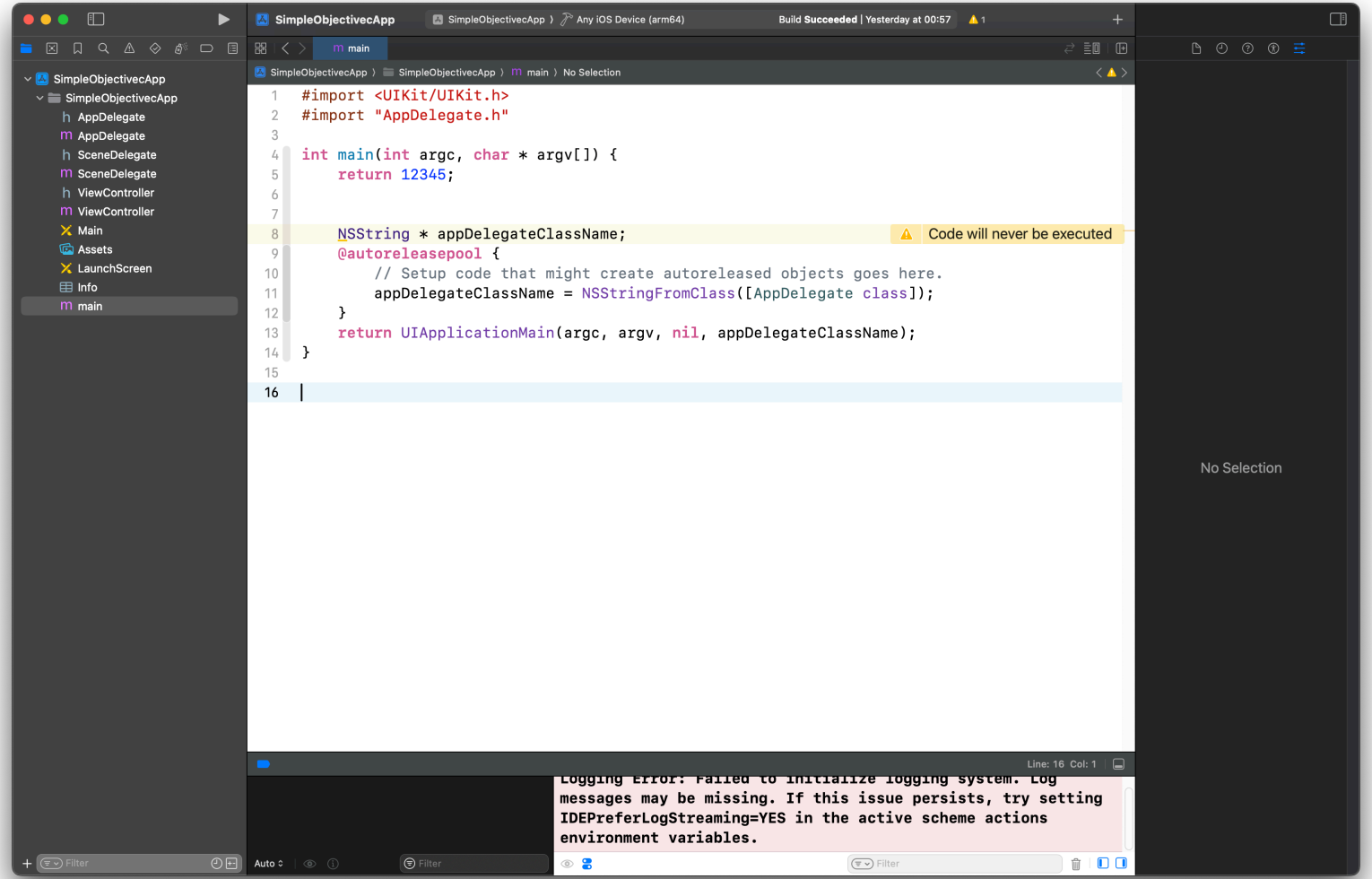
А как же iOS?

Simple app

Как выглядит минимальное приложение для iOS?


SimpleApp
SimpleApp ⟩ iPhone 15 Pro
SimpleApp: Ready | Today at 18:00
SimpleAppApp
SimpleApp ⟩ SimpleApp ⟩ SimpleAppApp ⟩ No Selection
SimpleApp
SimpleApp
SimpleAppApp
ContentView
Assets
Preview Content
Filter
import SwiftUI
@main
struct SimpleAppApp: App {
var body: some Scene {
WindowGroup {
ContentView()
}
}
}
Line: 11 Col: 1


SimpleApp
SimpleApp ⟩ iPhone 15 Pro
SimpleApp: Ready | Today at 18:00
Main (Base)
SimpleApp ⟩ SimpleApp ⟩ Main ⟩ Main (Base) ⟩ View Controller Scene ⟩ View Controller ⟩ View
SimpleApp
SimpleApp
AppDelegate
SceneDelegate
ViewController
Main
Assets
LaunchScreen
Info
View Controller Scene
View Controller
View
Safe Area
First Responder
Exit
Storyboard Entry Point
Filter
iPhone 15 Pro
71 %
Filter


SimpleObjectivecApp
SimpleObjectivecApp ⟩ Any iOS Device (arm64)
Build Succeeded | Yesterday at 00:57
main
SimpleObjectivecApp ⟩ SimpleObjectivecApp ⟩ main ⟩ No Selection
SimpleObjectivecApp
SimpleObjectivecApp
AppDelegate
AppDelegate
SceneDelegate
SceneDelegate
ViewController
ViewController
Main
Assets
LaunchScreen
Info
main
#import <UIKit/UIKit.h>
#import "AppDelegate.h"
int main(int argc, char * argv[]) {
return 12345;
NSString * appDelegateClassName;
Code will never be executed
@autoreleasepool {
// Setup code that might create autoreleased objects goes here.
appDelegateClassName = NSStringFromClass([AppDelegate class]);
}
return UIApplicationMain(argc, argv, nil, appDelegateClassName);
}
Line: 16 Col: 1
Logging Error: Failed to initialize logging system. Log messages may be missing. If this issue persists, try setting IDEPreferLogStreaming=YES in the active scheme actions environment variables.
No Selection

Что получим, если соберём?

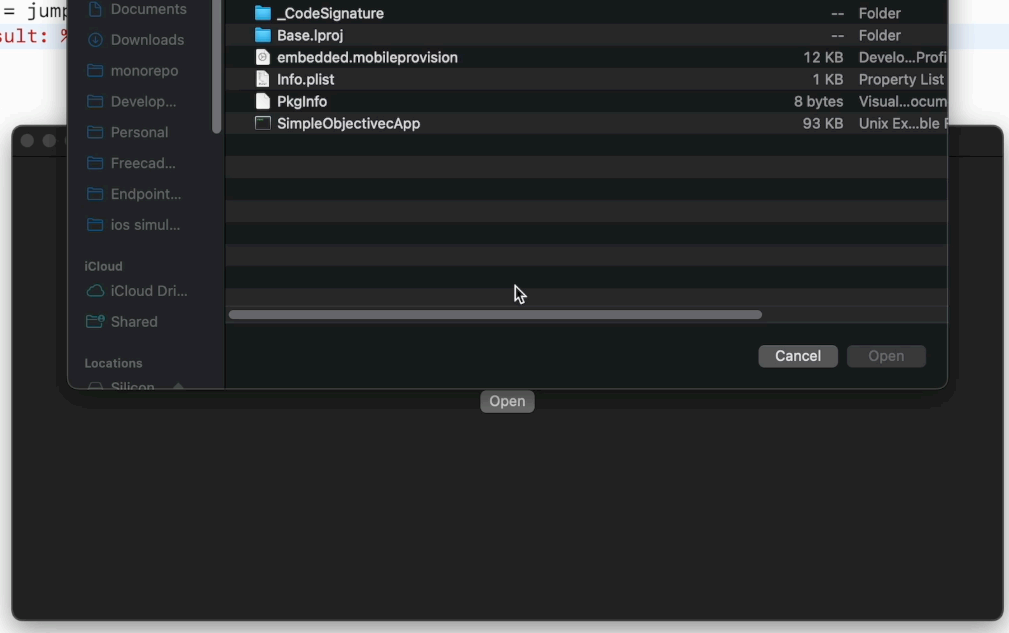
Полноценное приложение для iOS (ставится на устройство и запускается)

Не делает ничего, кроме как возвращает 12345

Нам для проверки хватит

Let’s do it [3]

```
10      int result = jump
11      NSLog(@"result: %
12  }
13
14  @end
15
```



Результаты

Запуск состоялся

Можно ли запускать что-то сложнее?

Нужно будет заимплементить “кое-что”

Кое-что:

Foundation

libSystem.dylib

libobjc.dylib

UIKit

Бонус

Запускаем macOS приложение на iOS

Почему должно работать?

Весь код писался на Swift / Objective-C++ / C

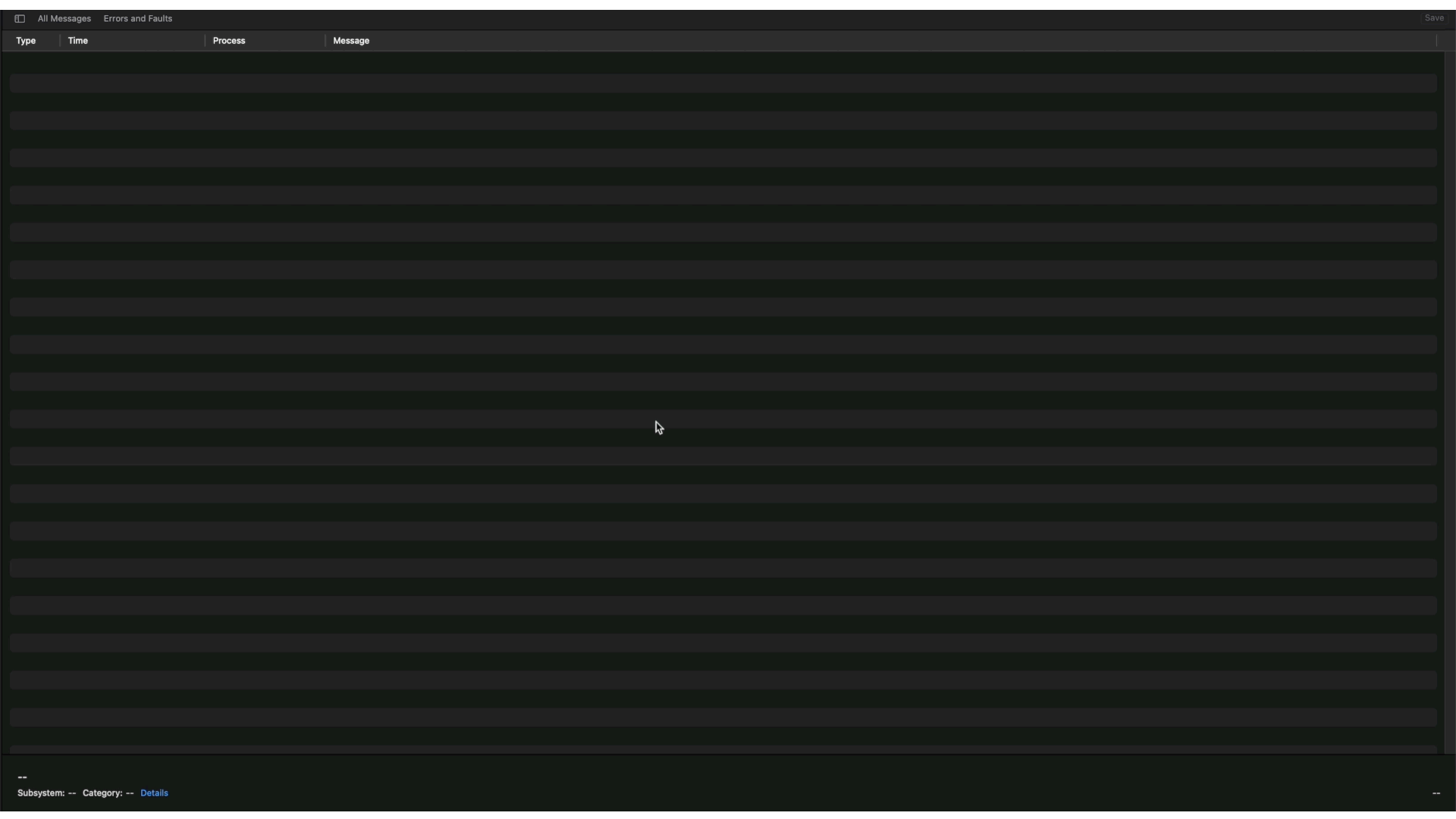
Зависимостей нет

UI на SwiftUI

Let’s do it [4]

Select file.

Open


All Messages
Errors and Faults
Save
Type
Time
Process
Message
--
Subsystem: -- Category: -- Details
--

Статистика

Сколько нужно порезолвить символов для загрузки libsystem?

7 752

Сколько нужно RAM для загрузки libsystem и зависимостей целиком?

350 MB

Сколько зависимостей у простейшего бинарника?

```
40
```

Сколько зависимостей у простейшего бинарника?

```
libsystem_blocks.dylib
libsystem_darwin.dylib
libsystem_sim_pthread_host.dylib
libcommonCrypto.dylib
libcorecrypto.dylib
libsystem_info.dylib
libsystem_c.dylib
libsystem_containermanager.dylib
libsystem_sanitizers.dylib
libsystem_sim_kernel_host.dylib
libsystem_notify.dylib
libremovefile.dylib
libsystem_eligibility.dylib
libdyld.dylib
libsystem_sim_platform_host.dylib
libsystem_pthread.dylib
libsystem_featureflags.dylib
libmacho.dylib
libxpc.dylib
libsystem_sim_kernel.dylib
libsystem_coreservices.dylib
libdispatch.dylib
libsystem_asl.dylib
libcopyfile.dylib
libsystem_collections.dylib
libsystem_sim_platform.dylib
libsystem_networkextension.dylib
libsystem_dnssd.dylib
libsystem_m.dylib
libsystem_platform.dylib
libsystem_sandbox.dylib
libsystem_kernel.dylib
libcache.dylib
libsystem_configuration.dylib
libunwind.dylib
libsystem_trace.dylib
libsystem_malloc.dylib
libcompiler_rt.dylib
libsystem_sim_pthread.dylib
libSystem.B.dylib
```

Сколько времени загружается наш бинарник?

10 секунд

Из приколов

Integer

```
struct dyld_chained_import
{
    uint32_t    lib_ordinal :  8,
                weak_import :  1,
                name_offset : 23;
};
```

n_sect

```
struct nlist_64 {
    union {
        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 */
    uint16_t n_desc;        /* see <mach-o/stab.h> */
    uint64_t n_value;       /* value of this symbol (or stab offset) */
};
```

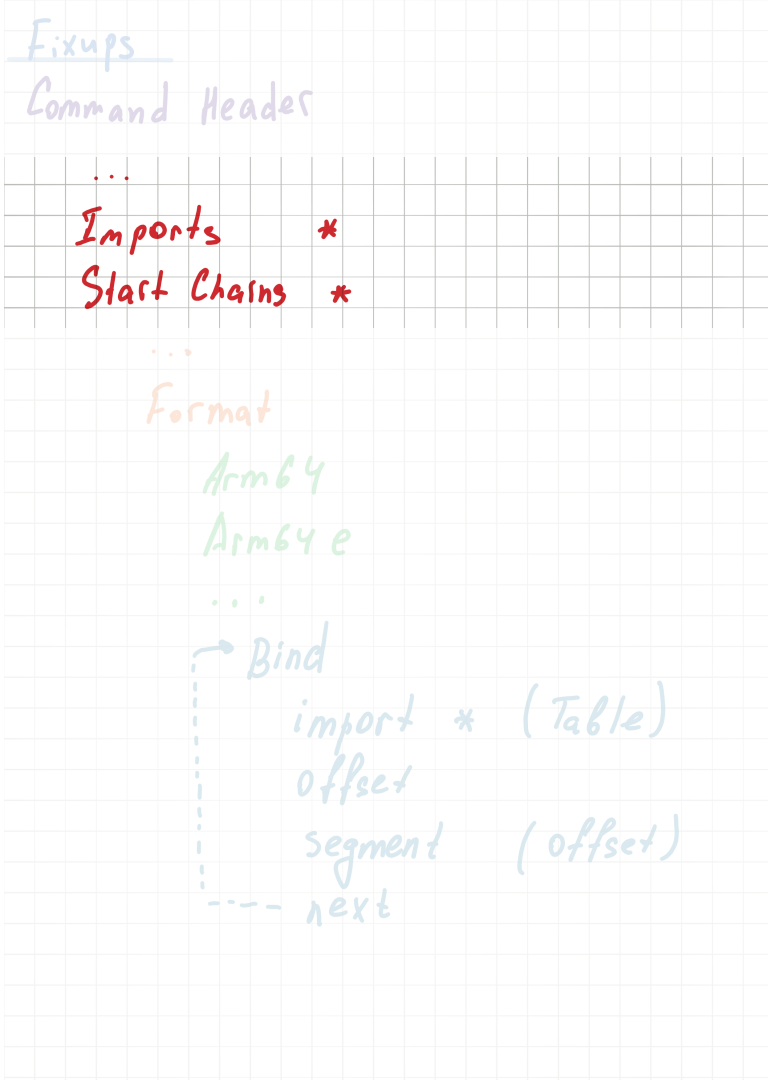
n_sect

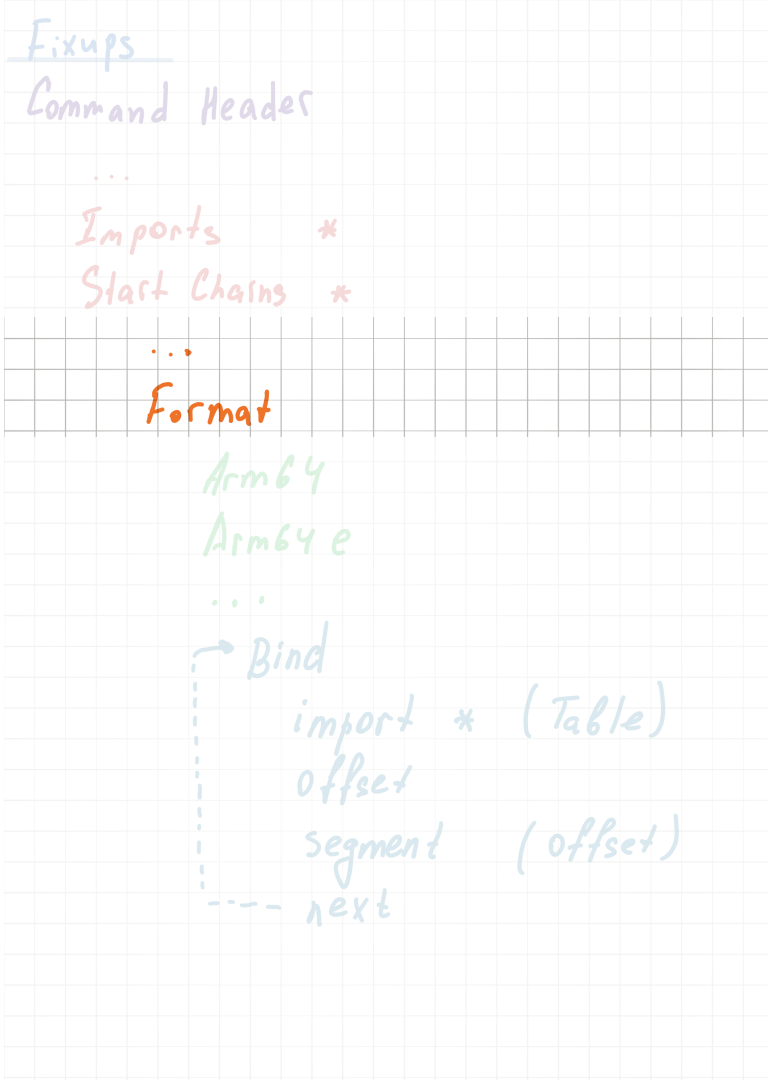
The n_sect value in the nlist_64 struct refers to the 1-based index of the section within the complete list of sections. The linker uses this index to locate the corresponding section by counting through the sections in all segments.

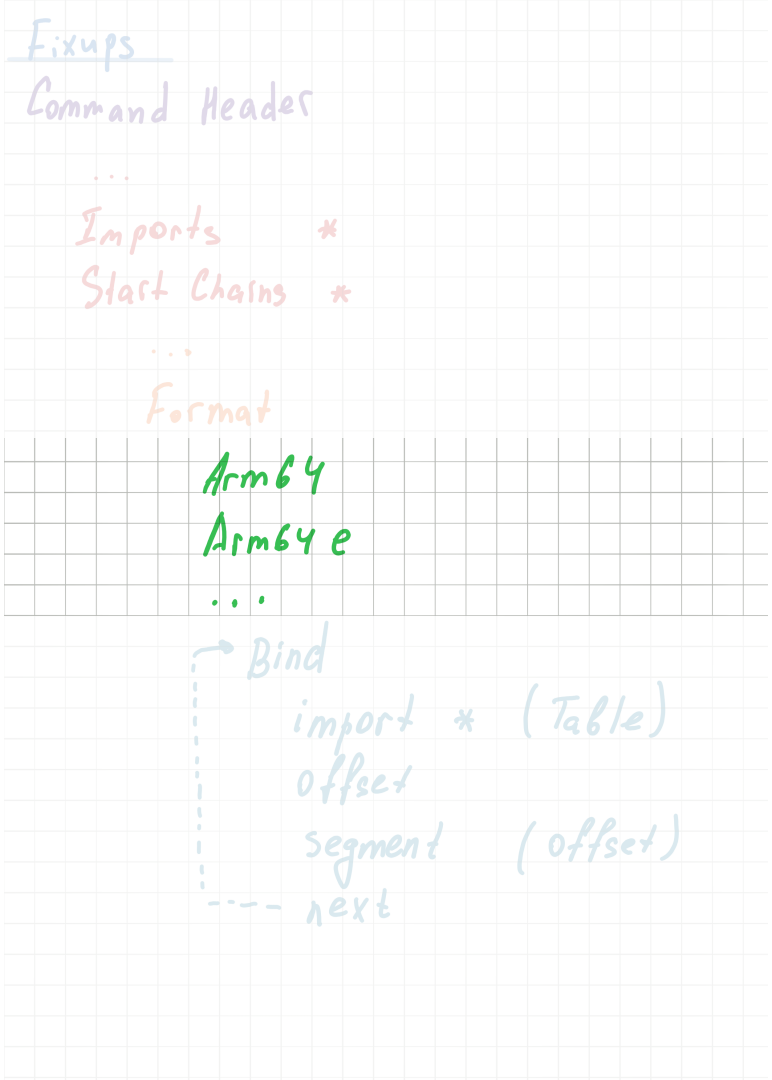
Fixups

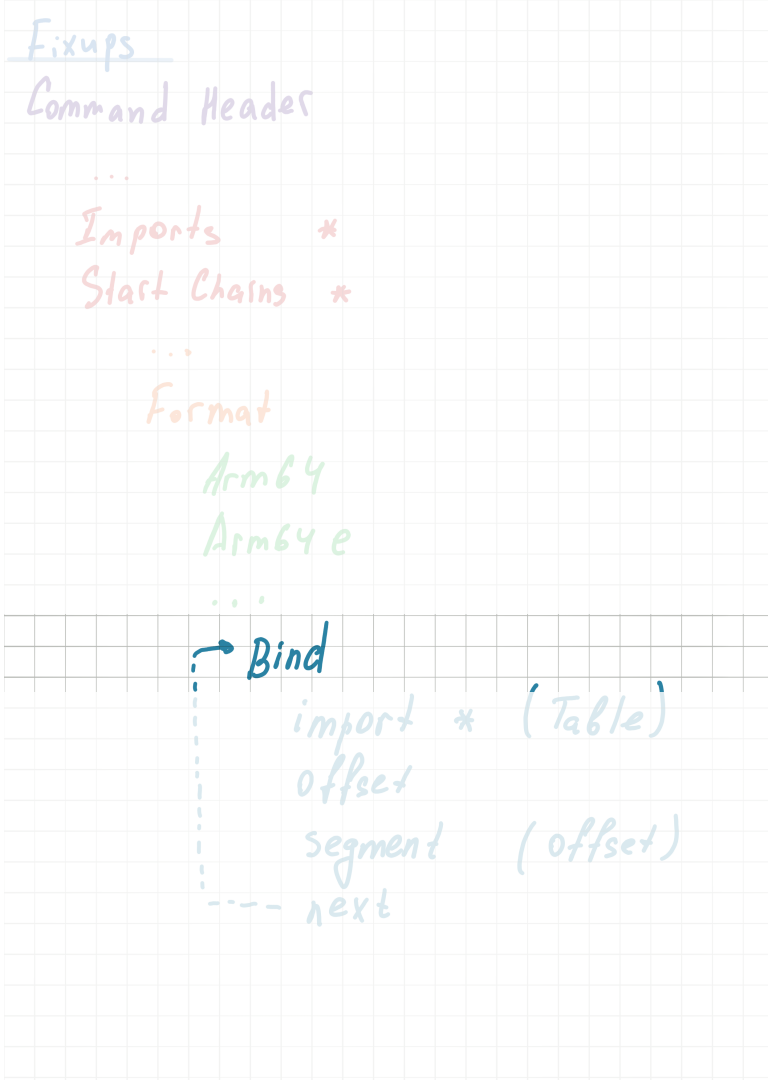
Fixups

Command Header


Fixups
Command Header
...
Imports *
Start Chains *
...
Format
Arm64
Arm64e
...
Bind
import * (Table)
offset
segment (offset)
next


Fixups
Command Header
...
Imports *
Start Chains *
...
Format
Arm64
Arm64e
...
Bind
import * (Table)
offset
segment (offset)
next


Fixups
Command Header
...
Imports *
Start Chains *
...
Format
Arm64
Arm64e
...
Bind
import * (Table)
offset
segment (offset)
next


Fixups
Command Header
...
Imports *
Start Chains *
...
Format
Arm64
Arm64e
...
Bind
import * (Table)
offset
segment (offset)
next

Fixups

Command Header

...

Imports *

Start Chains *

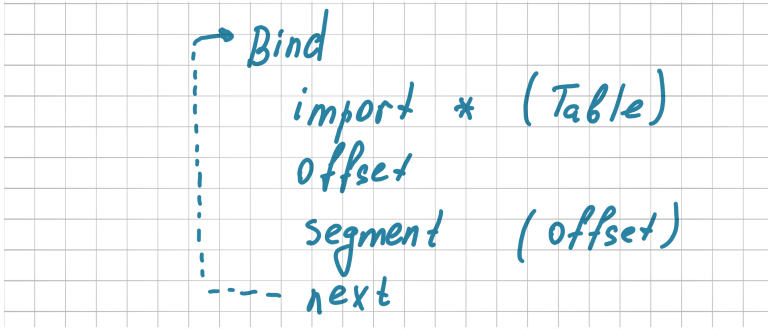
...

Format

Arm64

Arm64e

...



Fixups

Осталось только архивацию приделать…

Fixups

```
struct dyld_chained_fixups_header
{
    uint32_t    fixups_version;    // 0
    uint32_t    starts_offset;     // offset of dyld_chained_starts_in_image in chain_data
    uint32_t    imports_offset;    // offset of imports table in chain_data
    uint32_t    symbols_offset;    // offset of symbol strings in chain_data
    uint32_t    imports_count;     // number of imported symbol names
    uint32_t    imports_format;    // DYLD_CHAINED_IMPORT*
    uint32_t    symbols_format;    // 0 => uncompressed, 1 => zlib compressed
};
```

Итоги

Итоги

Плотно познакомились с Mach-O

Написали свой загрузчик

Научились работать с “сырой” памятью

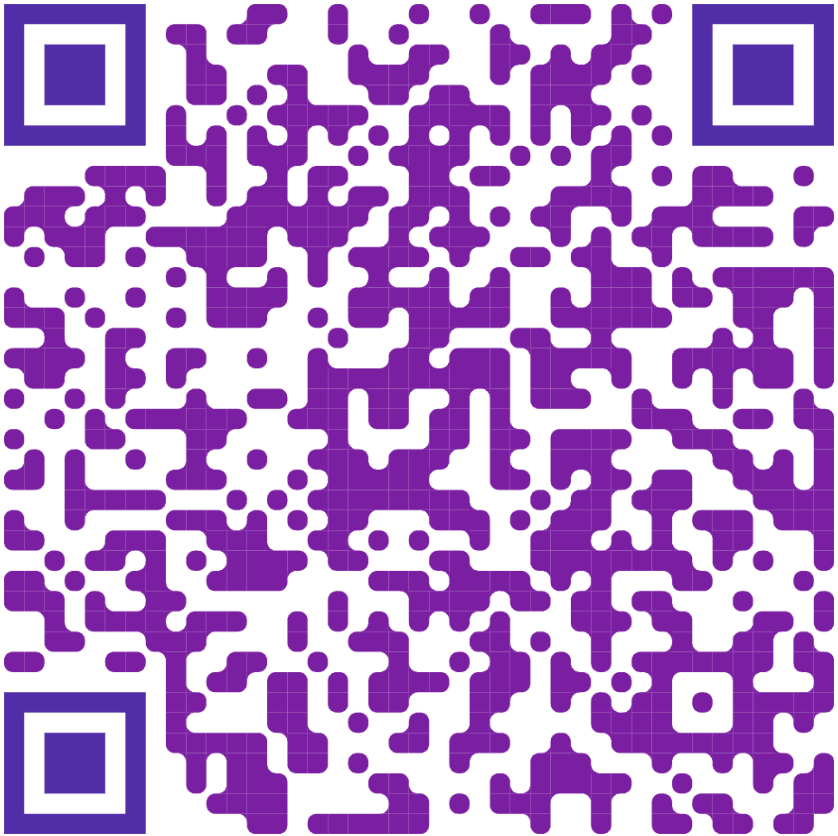
Немного поэксперементировали с подменой реализации

Итоги

Покопались в ASM / C коде

Запустили iOS app на macOS

Научились работать с инструментами (Ghidra / LLDB / Mach-O readers)



Проект на Github

Спасибо за внимание