# iOS Internals

# Kernel

The main kernel of iOS is the XNU kernel

A kernel can have many different Kexts or Kernel Extensions

These kernel extensions are usually not open source

Can be extracted with tools like joker

Kernel vs KernelCache ??

# Joker

```
prateek:./joker.universal -h
-h: No such file or directory
prateek:./joker.universal
Usage: joker [-j] [-MmaSsKk] _filename_
 _filename_ should be a decrypted iOS kernelcache, or kernel dump. Tested on ARMv7/s 3.x-9.3, and ARM64 through 11b9

 -m: dump Mach Traps and MIG tables (NEW)
 -a: dump everything
 -k: dump kexts
 -K: kextract [kext_bundle_id_or_name_shown_in_-k|all] to JOKER_DIR or /tmp
 -S: dump sysctls
 -s: dump UNIX syscalls
 -j: Jtool compatible output (to companion file) - 64bit kernels only

-dec: Decompress kernelcache to /tmp/kernel (complzss only at this stage)

Kernels not included. Get your own dump or decrypted kernel from iPhoneWiki, or Apple itself (as of iOS 10b1! Thanks, guys!)

4.0b with MACF Policies, stub symbolication, SPLIT KEXTS, no Sandbox Profiles (still beta, and AAPL enlarged profile again in
 11..) , kpp kernel zones(!) - and - IOUserClient methods!!
Compiled on Sep 10 2017

Contains code from Haruhiko Okumura (CompuServe 74050,1022) from BootX-81//bootx.tproj/sl.subproj/lzss.c
prateek:
```

```
prateek:./joker.universal  kernelcache.decrypted
mmapped: 0x126a85000
This is a 64-bit kernel from iOS 10.x, or later (3705.0.0.2.3)
ARM64 Exception Vector is at file offset @0x7b000 (Addr: 0xffffff00747f000)
prateek:
```

# Joker can extract kexts

```
0xffffff0053ac000: AppleSamsungPKE (com.apple.driver.AppleSamsungPKE)
0xffffff0053b8000: AppleInterruptController (com.apple.driver.AppleInterruptController)
0xffffff0053c4000: AppleAuthCP (com.apple.driver.AppleAuthCP)
0xffffff0053d0000: AppleS5L8960XGPIOIC (com.apple.driver.AppleS5L8960XGPIOIC)
0xffffff0053dc000: Seatbelt sandbox policy (com.apple.security.sandbox)
0xffffff0054680000: AppleHIDKeyboard (com.apple.driver.AppleHIDKeyboard)
0xffffff0054780000: AppleHDQGasGaugeControl (com.apple.driver.AppleHDQGasGaugeControl)
0xffffff00548c000: AppleAE2Audio (com.apple.driver.AppleAE2Audio)
0xffffff00549c000: AppleNANDConfigAccess (com.apple.driver.AppleNANDConfigAccess)
0xffffff0054a8000: IONVMeFamily (com.apple.iokit.IONVMeFamily)
0xffffff0054e8000: AppleDialogPMU (com.apple.driver.AppleDialogPMU)
0xffffff0054f8000: AppleD2255PMU (com.apple.driver.AppleD2255PMU)
0xffffff0055340000: I/O Kit HID Event Driver (com.apple.iokit.IOHIDEventDriver)
0xffffff0055340000: USBStorageDeviceSpecifics (com.apple.driver.USBStorageDeviceSpecifics)
0xffffff0055340000: IOAudioCodecs (com.apple.driver.IOAudioCodecs)
0xffffff0055700000: AppleDiskImagesReadWriteDiskImage (com.apple.driver.DiskImages.ReadWriteDiskImage)
0xffffff00557c000: AppleFSCompressionTypeZlib (com.apple.AppleFSCompression.AppleFSCompressionTypeZlib)
0xffffff0055880000: Broadcom 802.11 Driver (com.apple.driver.AppleBCMWLANCore)
0xffffff00568c000: AppleBCMWLANBusInterfacePCIe (com.apple.driver.AppleBCMWLANBusInterfacePCIe)
0xffffff0056bc000: AppleUSBHSIC (com.apple.driver.AppleUSBHSIC)
0xffffff0056cc000: Embedded I/O Kit Driver for USB EHCI Controllers (com.apple.driver.AppleUSBEHCIARM)
0xffffff0056dc000: AppleS5L8960XUSBHSIC (com.apple.driver.AppleS5L8960XUSBHSIC)
0xffffff0056e8000: AppleS5L8960XUSBEHCI (com.apple.driver.AppleS5L8960XUSBEHCI)
0xffffff0056f4000: AppleUSBDeviceNCM (com.apple.driver.AppleUSBDeviceNCM)
0xffffff0057000000: AppleBSDKextStarterVPN (com.apple.driver.AppleBSDKextStarterVPN)
0xffffff0057000000: AppleCS42L71Audio (com.apple.driver.AppleCS42L71Audio)
0xffffff0057180000: HFS (com.apple.filesystems.hfs.kext)
0xffffff00577c000: AppleS8000SmartIO (com.apple.driver.AppleS8000SmartIO)
0xffffff0057e8000: AppleSamsungI2S (com.apple.driver.AppleSamsungI2S)
0xffffff0057f4000: AppleM68Buttons (com.apple.driver.AppleM68Buttons)
0xffffff0058000000: I/O Kit Driver for USB HID Devices (com.apple.driver.usb.IOUSBHostHIDDeviceSafeBoot)
0xffffff0058000000: AppleUSBDeviceMux (com.apple.driver.AppleUSBDeviceMux)
0xffffff0058100000: PPTP (com.apple.nke.pptp)
0xffffff00581c000: AppleBasebandPCIMAVControl (com.apple.driver.AppleBasebandPCIMAVControl)
0xffffff0058480000: AppleS5L8960XWatchDogTimer (com.apple.driver.AppleS5L8960XWatchDogTimer)
0xffffff0058540000: AppleStorageDrivers (com.apple.driver.AppleStorageDrivers)
0xffffff0058540000: ApplePinotLCD (com.apple.driver.ApplePinotLCD)
0xffffff0058600000: IOAcceleratorFamily (com.apple.iokit.IOAcceleratorFamily)
0xffffff00588c000: I/O Kit Driver for USB User Clients (com.apple.iokit.usb.AppleUSBHostUserClient)
0xffffff00588c000: AppleUSBEthernetHost (com.apple.driver.AppleUSBEthernetHost)
0xffffff0058980000: AppleIDAMInterface (com.apple.driver.AppleIDAMInterface)
0xffffff0058a4000: I/O Kit HID Event Driver Safe Boot (com.apple.iokit.IOHIDEventDriverSafeBoot)
0xffffff0058a4000: AppleBasebandPCIMAVPDP (com.apple.driver.AppleBasebandPCIMAVPDP)
0xffffff0058b4000: AppleDiagnosticDataAccessReadOnly (com.apple.driver.AppleDiagnosticDataAccessReadOnly)
0xffffff0058c0000: AppleBiometricServices (com.apple.driver.AppleBiometricServices)
0xffffff0058cc000: AppleS5L8960XUSB (com.apple.driver.AppleS5L8960XUSB)
Got 190 kexts
prateek:
```

Important kexts

a) AppleMobileFileIntegrity.kext

b) Sandbox.kext

# Mandatory Access Control Framework (MACF)

Inherited from FreeBSD

Provides callouts for every user-controllable aspect of kernel functionality

Allows Kernel components to enforce any set of rules , a policy.

MACF doesn’t make decisions on its own, it leaves them to specialized kernel extensions.

# Mandatory Access Control Framework (MACF) continued

MACF is just a framework, doesn't provide any logic

Kernel extensions can register their interest with the framework for any operations that the framework intercepts

Once that function is called, appropriate arguments are sent to the extension

Decision is a boolean allow(0) or disallow(non-zero)

# Mandatory Access Control Framework (MACF) continued

A MACF policy defines a set of rules or conditions to be applied on a full or partial subset of kernel operation callouts.

An interested kernel extension can define and initialize a mac_policy_conf structure, and link to the MACF framework via a call mac_policy_register.

# Mandatory Access Control Framework (MACF) continued

```
        CHECK_SET_HOOK(iokit_check_get_property)
};

/*
 * Policy definition
 */
static struct mac_policy_conf policy_conf = {
        .mpc_name               = "CHECK",
        .mpc_fullname           = "Check Assumptions Policy",
        .mpc_field_off          = NULL,         /* no label slot */
        .mpc_labelnames         = NULL,         /* no policy label names */
        .mpc_labelname_count    = 0,            /* count of label names is 0 */
        .mpc_ops                = &policy_ops,  /* policy operations */
        .mpc_loadtime_flags     = 0,
        .mpc_runtime_flags      = 0,
};

static mac_policy_handle_t policy_handle;

/*
 * Init routine; for a loadable policy, this would be called during the KEXT
 * initialization; we're going to call this from bsd_init() if the boot
 * argument for checking is present.
 */
```

# Apple Mobile File Integrity (AMFI)

AppleMobileFileIntegrity(.kext), which can go by its full name com.apple.driver.AppleMobileFileIntegrity, is an iOS kernel extension which serves as the corner stone of iOS's code entitlements model. It is one of the Sandbox's (com.apple.security.sandbox) dependencies, along with com.apple.kext.AppleMatch (which, like on OS X, is responsible for parsing the Sandbox language rules).

This kext recognizes the task_for_pid-allow entitlement (among others) and is responsible for hooking this Mach call, which retrieves the Mach task port associated with a BSD process identifier. Given this port, one can usurp control of the task/PID, reading and writing its memory, debugging, etc. It is therefore enabled only if the binary is digitally signed with a proper entitlement file, specifying task_for_pid-allow.

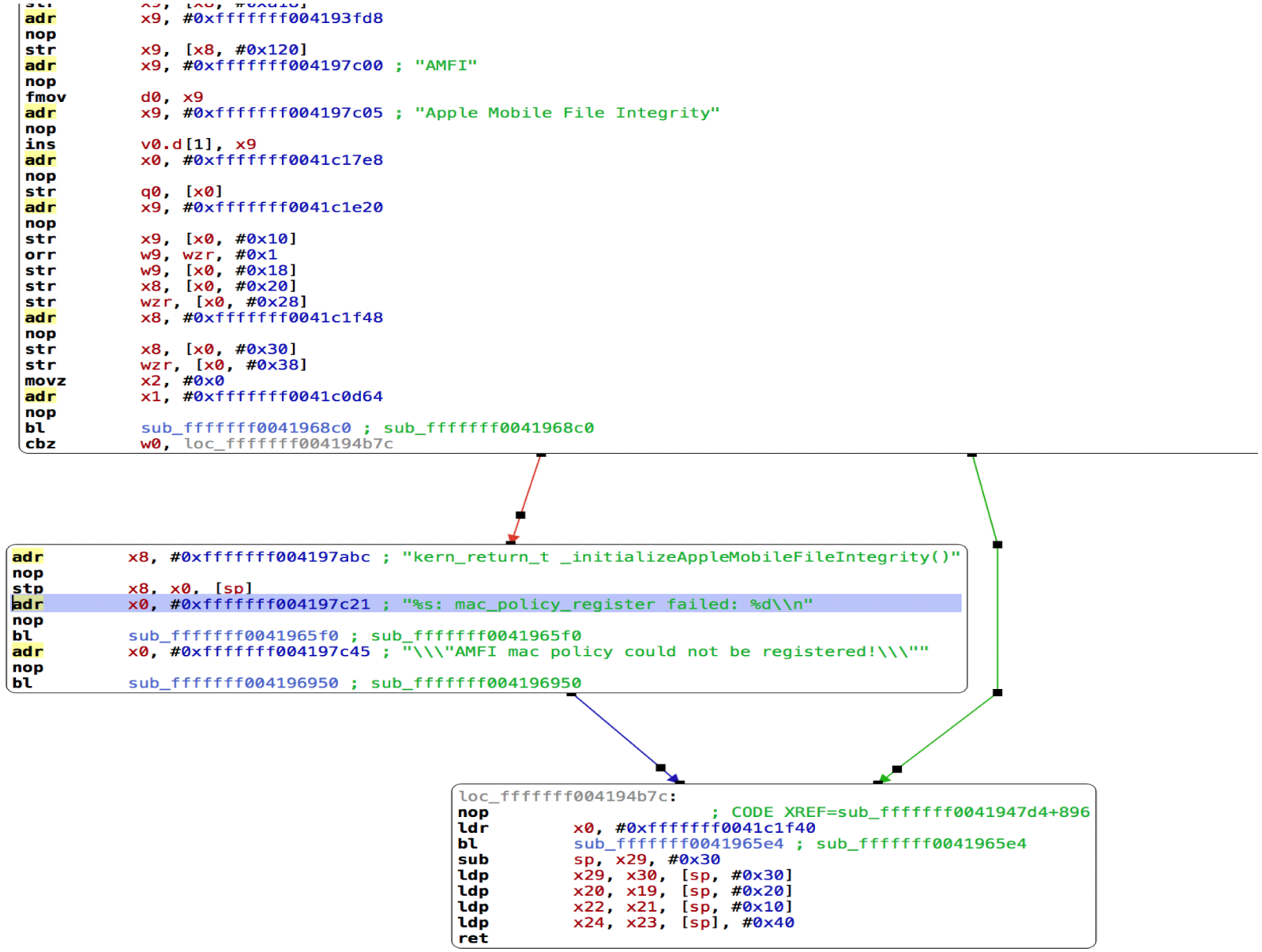
# AMFI policy register, extract AMFI

```
Got 190 kexts
prateek:./joker.universal -K com.apple.driver.AppleMobileFileIntegrity kernelcache.decrypted
mmapped: 0x1279a3000
This is a 64-bit kernel from iOS 10.x, or later (3705.0.0.2.3)
ARM64 Exception Vector is at file offset @0x7b000 (Addr: 0xffffffff00747f000)
Found com.apple.driver.AppleMobileFileIntegrity at load address: ffffffff004190000, offset: 700000
Writing kext out to /tmp/com.apple.driver.AppleMobileFileIntegrity.kext
Unable to resolve kernel symbol at ffffffff0041550d4 (this is fine if it's a symbol from another kext)
Unable to resolve kernel symbol at ffffffff004142c6c (this is fine if it's a symbol from another kext)
Unable to resolve kernel symbol at ffffffff0041711f0 (this is fine if it's a symbol from another kext)
Unable to resolve kernel symbol at ffffffff00416b834 (this is fine if it's a symbol from another kext)
Warning: Disassembly left some unhandled instructions!
Symbolicated stubs to /tmp/com.apple.driver.AppleMobileFileIntegrity.kext.ARM64.C4CEB8BB-3F4C-31B0-B376-9823F99FE031
Dumping symbol cache to file
prateek:open /tmp/
prateek:
```

# AMFI policy register

```
fffffff004197c00        db      "AMFI", 0                               ; DATA XREF=sub_fffffff0041947d4+800
                        aAppleMobileFil:
fffffff004197c05        db      "Apple Mobile File Integrity", 0        ; DATA XREF=sub_fffffff0041947d4+812
                        aSMacpolicyregi:
fffffff004197c21        db      "%s: mac_policy_register failed: %d\n", 0   ; DATA XREF=sub_fffffff0041947d4+912
                        aAmfiMacPolicyC:
fffffff004197c45        db      "\"AMFI mac policy could not be registered!\"", 0 ; DATA XREF=sub_fffffff0041947d4+924
                        aImgvp:
fffffff004197c70        db      "img_vp", 0                             ; DATA XREF=sub_fffffff004195638+528
                        aAmfiAllowingPi:
fffffff004197c77        db      "AMFI: allowing pid %u to inherit IPC ports (platform binary)\n", 0 ; DATA XREF=sub_fffffff004195638+180
```

# AMFI policy register



# Entitlements

Entitlements are flags that become part of your signed app. These allow you access to various things like APNS, iCloud, Siri, Apple Pay etc. (Your provisioning profile needs to match, otherwise your app won't start.)

Entitlements are a way of your app saying "I want to use X feature" where X might be APNS. Your provision profile has to match that and say, "The app can use feature X". Since entitlements are part of the code-signed app, it's a verification that the app is qualified and allowed to use that feature.

Uber app had a special feature to look for screen recording apps.

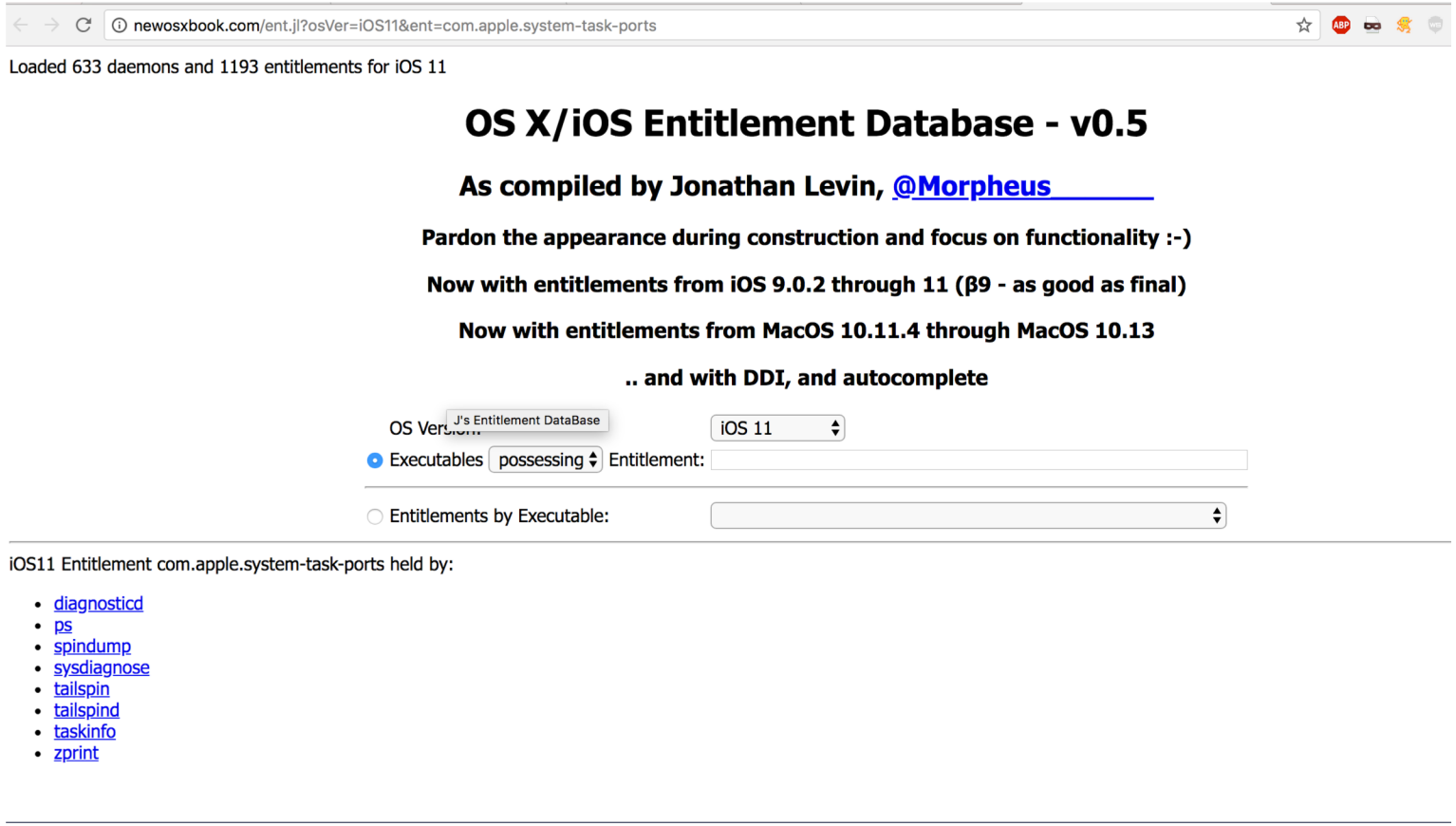
# Entitlements

```
prateek:/Users/prateek/Downloads/jtool/jtool --ent /Applications/Xcode.app/Contents/MacOS/Xcode
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>com.apple.PairingManager.Read</key>
        <true/>
        <key>com.apple.PairingManager.Write</key>
        <true/>
        <key>com.apple.application-identifier</key>
        <string>59GAB85EFG.com.apple.dt.Xcode</string>
        <key>com.apple.authkit.client.private</key>
        <true/>
        <key>com.apple.developer.maps</key>
        <true/>
        <key>com.apple.private.coreservices.definesExtensionPoint</key>
        <true/>
        <key>com.apple.sysmond.client</key>
        <true/>
</dict>
</plist>

prateek:
```

# Entitlements

```
Prateek:~ prateekg147$ jtool --ent /bin/ps
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>com.apple.system-task-ports</key>
        <true/>
        <key>task_for_pid-allow</key>
        <true/>
</dict>
</plist>

Prateek:~ prateekg147$
```

# J’s entitlement database



# Code Signing

The origin of the code – Verified with public key confirms that it was signed with private key

Authenticity of the code – Any modification would break the digital signature

Integration with entitlements makes it hard to defeat code signing

Apple extended Mach-O format to add the LC_CODE_SIGNATURE load command

Code signature is attached to the very last

CDHash is the mega hash of all code directory hashes

# Code Signing

```
                        ;
                        ; Load Command 35
                        ;
000050b0        struct __macho__linkedit_data_command {
                    LC_DATA_IN_CODE,                    // LC_CODE_SIGNATURE, LC_SEGMENT_SPLIT_INFO, LC_FUNCTION_STARTS, LC_DATA_IN_C
                    0x10,                               // sizeof(struct linkedit_data_command)
                    0x5af700,                           // file offset of data in __LINKEDIT segment
                    0x8838                              // file size of data in __LINKEDIT segment
                }
                        ;
                        ; Load Command 36
                        ;
000050c0        struct __macho__linkedit_data_command {
                    LC_DYLIB_CODE_SIGN_DRS,             // LC_CODE_SIGNATURE, LC_SEGMENT_SPLIT_INFO, LC_FUNCTION_STARTS, LC_DATA_IN_C
                    0x10,                               // sizeof(struct linkedit_data_command)
                    0x5b7f38,                           // file offset of data in __LINKEDIT segment
                    0xbc                                // file size of data in __LINKEDIT segment
                }
                        ;
                        ; Load Command 37
                        ;
000050d0        struct __macho__linkedit_data_command {
                    LC_CODE_SIGNATURE,                  // LC_CODE_SIGNATURE, LC_SEGMENT_SPLIT_INFO, LC_FUNCTION_STARTS, LC_DATA_IN_C
                    0x10,                               // sizeof(struct linkedit_data_command)
                    0x89e730,                           // file offset of data in __LINKEDIT segment
                    0xad00                              // file size of data in __LINKEDIT segment
                }
000050e0        db         22416 dup (0x00)
```

# Code Signing

```
          No modtab
          159 Indirect symbols at offset 0x6ba0

LC 07: LC_LOAD_DYLINKER          /usr/lib/dyld
LC 08: LC_UUID                   UUID: BDFAAB38-B160-30D2-8549-B94AFD2F1B1F
LC 09: LC_VERSION_MIN_MACOSX     Minimum OS X version:    10.13.0
LC 10: LC_SOURCE_VERSION         Source Version:          272.0.0.0.0
LC 11: LC_MAIN                   Entry Point:             0x1200 (Mem: 0x10000120
0)
LC 12: LC_LOAD_DYLIB             /usr/lib/libutil.dylib
LC 13: LC_LOAD_DYLIB             /usr/lib/libncurses.5.4.dylib
LC 14: LC_LOAD_DYLIB             /usr/lib/libSystem.B.dylib
LC 15: LC_FUNCTION_STARTS        Offset: 26112, Size: 56 (0x6600-0x6638)
LC 16: LC_DATA_IN_CODE           Offset: 26168, Size: 40 (0x6638-0x6660)
LC 17: LC_CODE_SIGNATURE         Offset: 29168, Size: 9520 (0x71f0-0x9720)
Prateek:~ prateekg147$ jtool -l /bin/ls | grep CODE_SIG
LC 17: LC_CODE_SIGNATURE         Offset: 29168, Size: 9520 (0x71f0-0x9720)
Prateek:~ prateekg147$
```

# Code Signing

```
Prateek:~ prateekg147$ jtool -l /bin/ls | grep CODE_SIG
LC 17: LC_CODE_SIGNATURE        Offset: 29168, Size: 9520 (0x71f0-0x9720)
Prateek:~ prateekg147$ dd if=/bin/ls of =
Prateek:~ prateekg147$ dd if=/bin/ls of=ls.sig bs=29168 skip=1
0+1 records in
0+1 records out
9520 bytes transferred in 0.000313 secs (30434279 bytes/sec)
Prateek:~ prateekg147$ file ls.sig
ls.sig: Mac OS X Detached Code Signature (non-executable) - 4970 bytes
Prateek:~ prateekg147$ hexdump -C ls.sig
00000000  fa de 0c c0 00 00 13 6a  00 00 00 03 00 00 00 00  |.......j........|
00000010  00 00 00 24 00 00 00 02  00 00 01 a1 00 01 00 00  |...$............|
00000020  00 00 01 dd fa de 0c 02  00 00 01 7d 00 02 01 00  |...........}....|
00000030  00 00 00 00 00 00 00 7d  00 00 00 30 00 00 00 02  |.......}...0....|
00000040  00 00 00 08 00 00 71 f0  20 02 04 0c 00 00 00 00  |......q. .......|
00000050  00 00 00 00 63 6f 6d 2e  61 70 70 6c 65 2e 6c 73  |....com.apple.ls|
00000060  00 a8 cc c6 0c 2a 5b ff  15 80 5b eb 86 87 c6 a8  |.....*[...[.....|
```

# Code Signing

Signing the whole binary can be a costly operation

Each binary page is hashed individually

SHA 256 being used since iOS 10, SHA1 deprecated

By default, code signing only in the text segment, not data segment. Hence data only attacks are very common

# Code Signing

```
Prateek:~ prateekg147$ jtool --pages /bin/ls
0x0-0x5000      __TEXT
        0xf20-0x442e    __TEXT.__text
        0x442e-0x45f6   __TEXT.__stubs
        0x45f8-0x4900   __TEXT.__stub_helper
        0x4900-0x4af0   __TEXT.__const
        0x4af0-0x4f69   __TEXT.__cstring
        0x4f6c-0x4ffc   __TEXT.__unwind_info
0x5000-0x6000   __DATA
        0x5000-0x5028   __DATA.__got
        0x5028-0x5038   __DATA.__nl_symbol_ptr
        0x5038-0x5298   __DATA.__la_symbol_ptr
        0x52a0-0x54c8   __DATA.__const
        0x54d0-0x54f8   __DATA.__data
0x6000-0x9720   __LINKEDIT
        0x6000-0x6018   Rebase Info     (opcodes)
        0x6018-0x6080   Binding Info    (opcodes)
        0x6080-0x65e0   Lazy Bind Info  (opcodes)
        0x65e0-0x6600   Exports
        0x6600-0x6638   Function Starts
        0x6638-0x6660   Data In Code
        0x6660-0x6ba0   Symbol Table
        0x6ba0-0x6e1c   Indirect Symbol Table
        0x6e1c-0x71ec   String Table
        0x71f0-0x9720   Code Signature
Prateek:~ prateekg147$
```

# Code Signing – Special slots

```
Prateek:~ prateekg147$ jtool --sig -v /Applications/Mail.app | head -14
Blob at offset: 5100208 (52416 bytes) is an embedded signature of 47861 bytes, and 4 blobs
        Blob 0: Type: 0 @44: Code Directory (40095 bytes)
                Version:     20100
                Flags:       none (0x0)
                Platform Binary
                CodeLimit:   0x4dd2b0
                Identifier:  com.apple.mail (0x30)
                CDHash:      db632e7157ab22289979265f59406365f664e1c63002a9d16c323b1255110
509 (computed)
                # of Hashes: 1246 code + 5 special
                Hashes @223 size: 32 Type: SHA-256
                         Entitlements blob:      c02cbdbda2433a3bd6d2d22fe47617faaabfbeff68
dcea856c0599a7b5639482 (OK)
                         Application Specific:   Not Bound
                         Resource Directory:     c29b905b0a472d7c326f5a0c62acdadb4de237575c
7de24cc610354dbf5ceaf0 (OK)
                         Requirements blob:      6d1b0aedac9497f4314ef726c08f36faea3bf7a5e8
a5f7ee836f8bb4429852e4 (OK)
Prateek:~ prateekg147$
```

# Apple Mobile File Integrity (AMFI)

AMFI enforces code signing throughout the OS

It has a built in cache with CDHashes of all iOS binaries

For all other binaries (including iOS apps), it uses amfid

# Apple Mobile File Integrity (AMFI)

```
Prateek:Desktop prateekg147$ jtool --sig amfid | grep CDHash
                CDHash:      87100d66435fadf19c87e7de59964db494703ecc (computed)
Prateek:Desktop prateekg147$
```

```
0002b3f0  2b 06 01 05 05 07 02 02  55 04 03 2a 86 48 86 f7  |+.......U..*.H..|
0002b400  0d 01 09 01 55 04 0a 2a  86 48 86 f7 63 64 06 01  |....U..*.H..cd..|
0002b410  1c 2a 86 48 86 f7 63 64  06 01 1c 01 2a 86 48 86  |.*.H..cd....*.H.|
0002b420  f7 63 64 06 02 0a 87 10  0d 66 43 5f ad f1 9c 87  |.cd......fC_....|
0002b430  e7 de 59 96 4d b4 94 70  3e cc 00 00 00 00 00 00  |..Y.M..p>.......|
0002b440  00 00 00 00 00 00 00 00  00 00 00 00 00 00 4c 69  |..............Li|
0002b450  62 72 61 72 79 20 56 61  6c 69 64 61 74 69 6f 6e  |brary Validation|
0002b460  20 6f 76 65 72 72 69 64  64 65 6e 20 66 6f 72 20  | overridden for |
0002b470  27 25 73 27 20 28 54 65  61 6d 20 49 44 3a 20 25  |'%s' (Team ID: %|
0002b480  73 2c 20 70 6c 61 74 66  6f 72 6d 3a 20 25 73 29  |s, platform: %s)|
0002b490  20 66 6f 72 20 70 72 6f  63 65 73 73 20 27 25 73  | for process '%s|
0002b4a0  28 25 64 29 27 20 28 54  65 61 6d 20 49 44 3a 20  |(%d)' (Team ID: |
```

THANK YOU !!