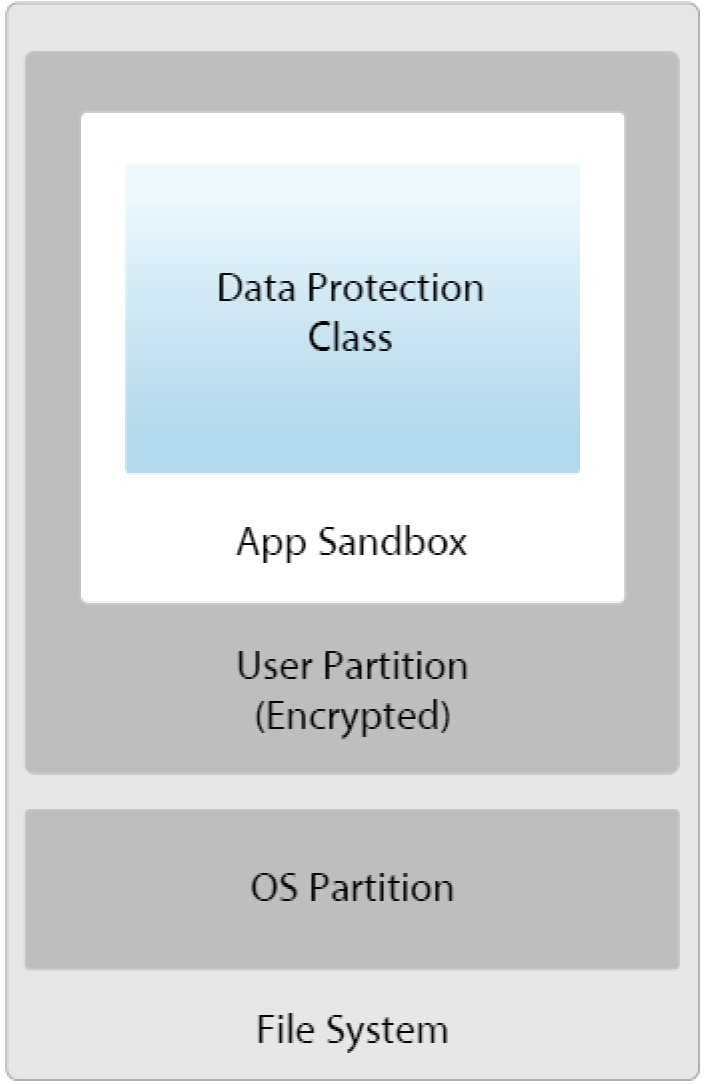
iOS Application Security

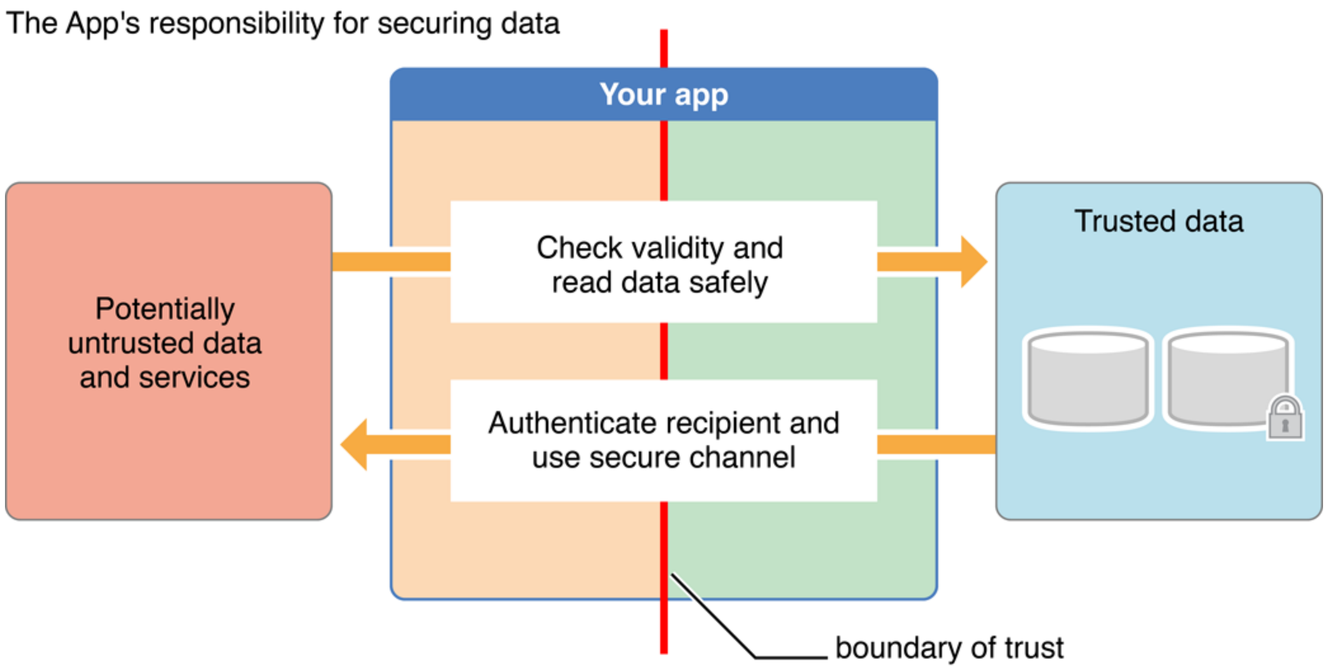
ACN / Mobile Security 2020

Johannes Feichtner
johannes.feichtner@iaik.tugraz.at

Outline

- App-Level Security on iOS
  - (Real) Code Signing
  - Sandbox
- App Internals
- App Analysis on iOS
  - Case Studies with Real Apps





Source: https://goo.gl/8X11Rf

APP STALKING —

Dozens of iOS apps surreptitiously share user location data with tracking firms

Applications don't mention that they're selling your precise location to third parties.

SEAN GALLAGHER - 9/10/2018, 9:11 PM

Source: https://goo.gl/FjCesH

What?

Location data of popular apps leaked to 12 known monetarization firms

- Bluetooth LE Beacon Data
- GPS Longitude and Latitude
- Wi-Fi SSID (Network Name) and BSSID (Network MAC Address)
- Further device data
  - Accelerometer, Cell network MCC/MNC, Battery Charge % and status (Battery or charged via USB)

Problem?

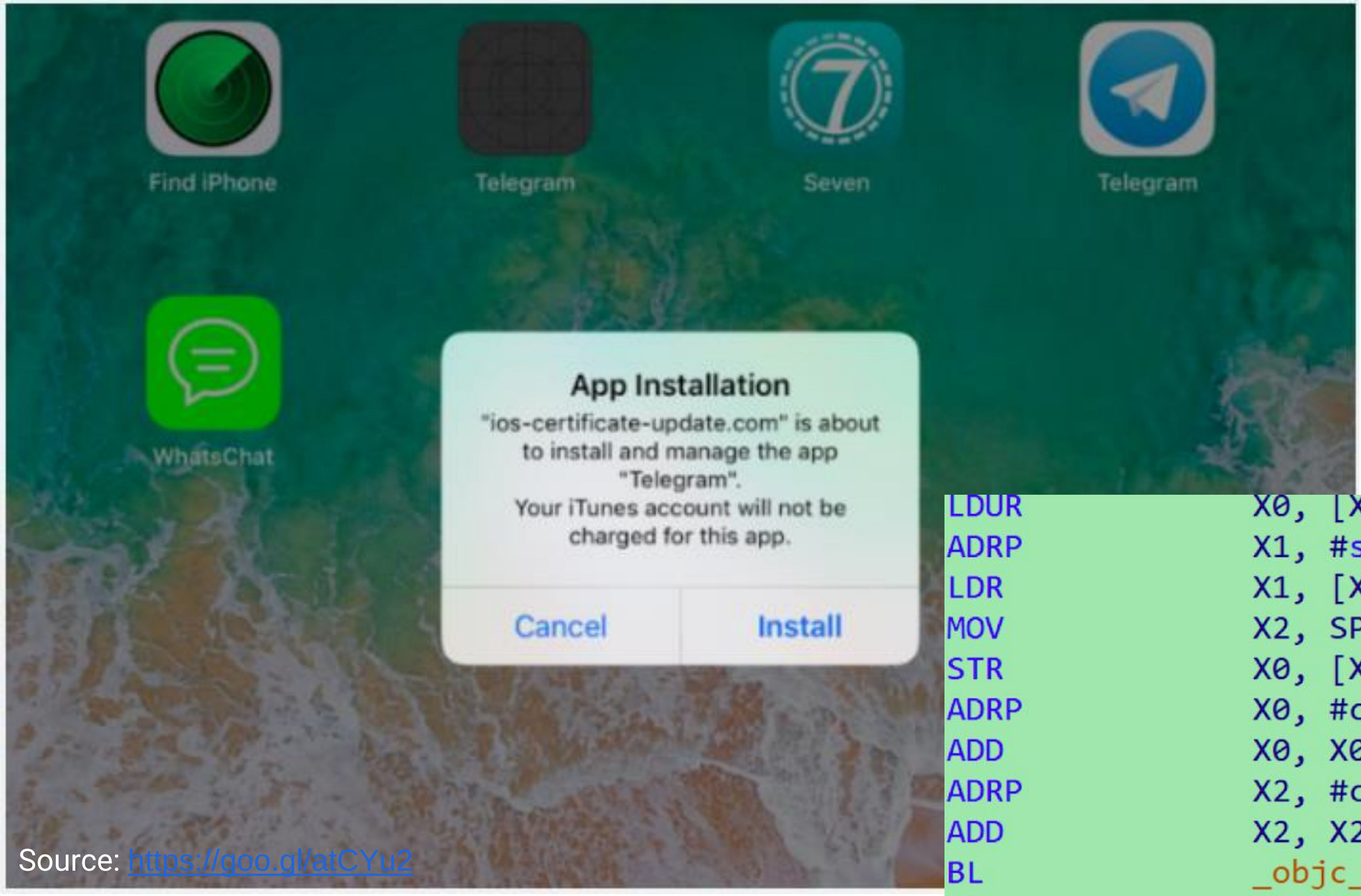
Users *agree* on sharing their location for different purposes, e.g. „Location based social networking for meeting people nearby"

FREE HACKS INSIDE —

Hyper-targeted attack against 13 iPhones dropped malicious apps via MDM

Installed hacked versions of Telegram, WhatsApp, and tracked users' location and SMS.

SEAN GALLAGHER - 7/13/2018, 5:47 PM



Source: https://goo.gl/atCYu2

What?

13 devices enrolled to attacker-controlled MDM server after physical access or via social engineering

Problem?

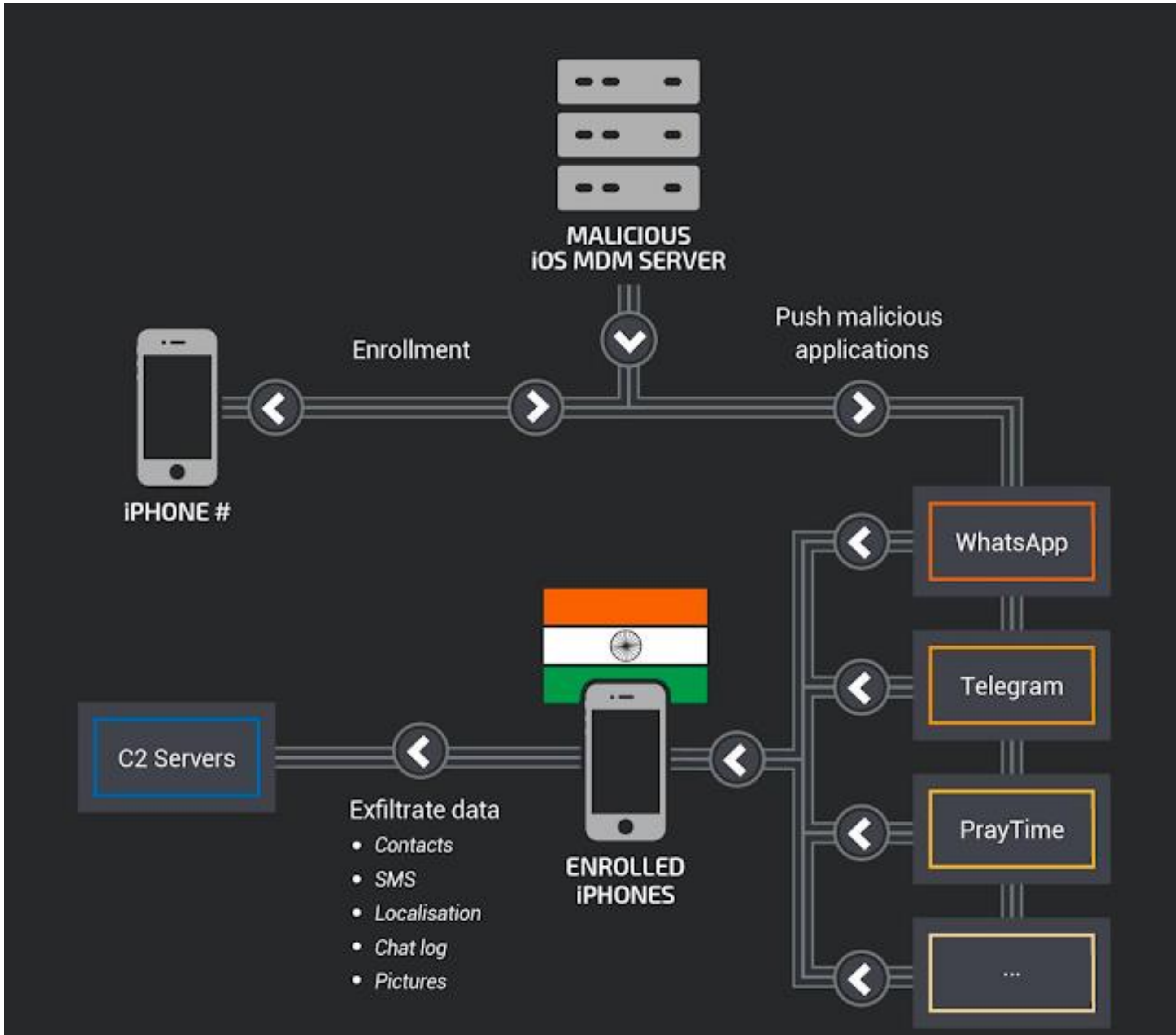
- MDM enrollment brought certificate → Trust to apps signed by third-party
- Inject code into messenger apps
- Upload to attacker server

Source: https://goo.gl/d6V67E

```
LDUR            X0, [X29,#location]
ADRP            X1, #selRef_stringByAppendingFormat_@PAGE
LDR             X1, [X1,#selRef_stringByAppendingFormat_@PAGEOFF] ; SEL
MOV             X2, SP
STR             X0, [X2,#0x40+var_40]
ADRP            X0, #cfstr_HttpTechwachCo@PAGE ; "http://techwach.com/Reduce/"
ADD             X0, X0, #cfstr_HttpTechwachCo@PAGEOFF ; id
ADRP            X2, #cfstr_Php@PAGE ; "%@.php"
ADD             X2, X2, #cfstr_Php@PAGEOFF ; "%@.php"
BL              _objc_msgSend
MOV             X29, X29
```

How?

1. User visits MDM web frontend
   – http://ios-certificate-update.com
   – http://www.wpitcher.com
2. Device enrolment with user interaction
   – Certificate authority installed
   – MDM has full control over device
3. Use BOptions sideloading technique to inject dynamic lib into legitimate app
   – Malware in custom BOptionspro.dylib
   – Bundled with original iOS app
   – Lib can ask for more permissions, execute code, steal info from original app



Source: https://goo.gl/d6V67E

→ Backdoor code to read/send data from WhatsApp, Telegram, … databases to C2 server http://techwach.com

App-Level Security

IAIK TU Graz

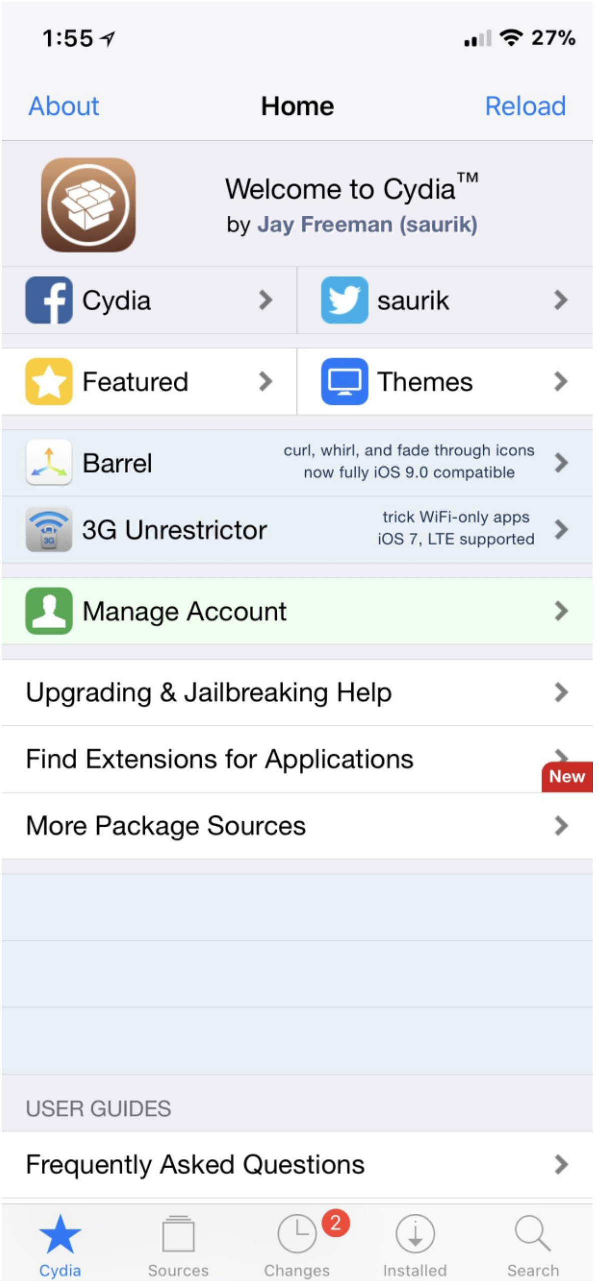
Installing iOS Apps

Officially…

- Via Apple App Store
  - Pre-installed on all iDevices
  - Only manually reviewed apps!
  - Developer‘s identities are verified by Apple
- Enterprise Mobile Device Management
- Sideloading
  - Signing app with developer certificate
  - Install / „trust“ developer certificate on device via Xcode

With Jailbreak

- Via file system
- Cydia package manager



Apple App Store

Review process

1. Developer uploads app
2. Enter queue for manual review (on re-upload: back to start)
3. Enter review in progress
   – On reject: Notification with reason
   – On success: App release

- 40 reviewers in 2009, each app with >= 2 reviews http://goo.gl/NSthWH
- Focus on bugs, instabilities, privacy violations, censorship, …
- Details about security checks not known

\+ Quality control and nearly no evil apps
\- Not possible to fix bugs / security issues quickly

Code Signing

All binaries and libraries must be signed!

- Or phone is specially provisioned
- Main reason why apps have to come from official store
- Signing certificates trusted on every device
- Trust Chain with Intermediate & Root CAs stored in OS

How to verify signatures?

1. Get team ID from certificate
2. Check if used libraries & app binary match signature
3. Linking with same signature as executable always possible

IAIK TU Graz

Code Signing Enforcement

When?

- Upon app or binary execution (= at runtime)
- Process may only execute if signed with valid & trusted signature

Security implications

- Ensures that process stays dynamically valid
  - No introduction of new executable code
  - Existing executable code cannot be changed
- Guarantees that running app == reviewed app
- Prevents code injection (no memory pages are writable & executable)

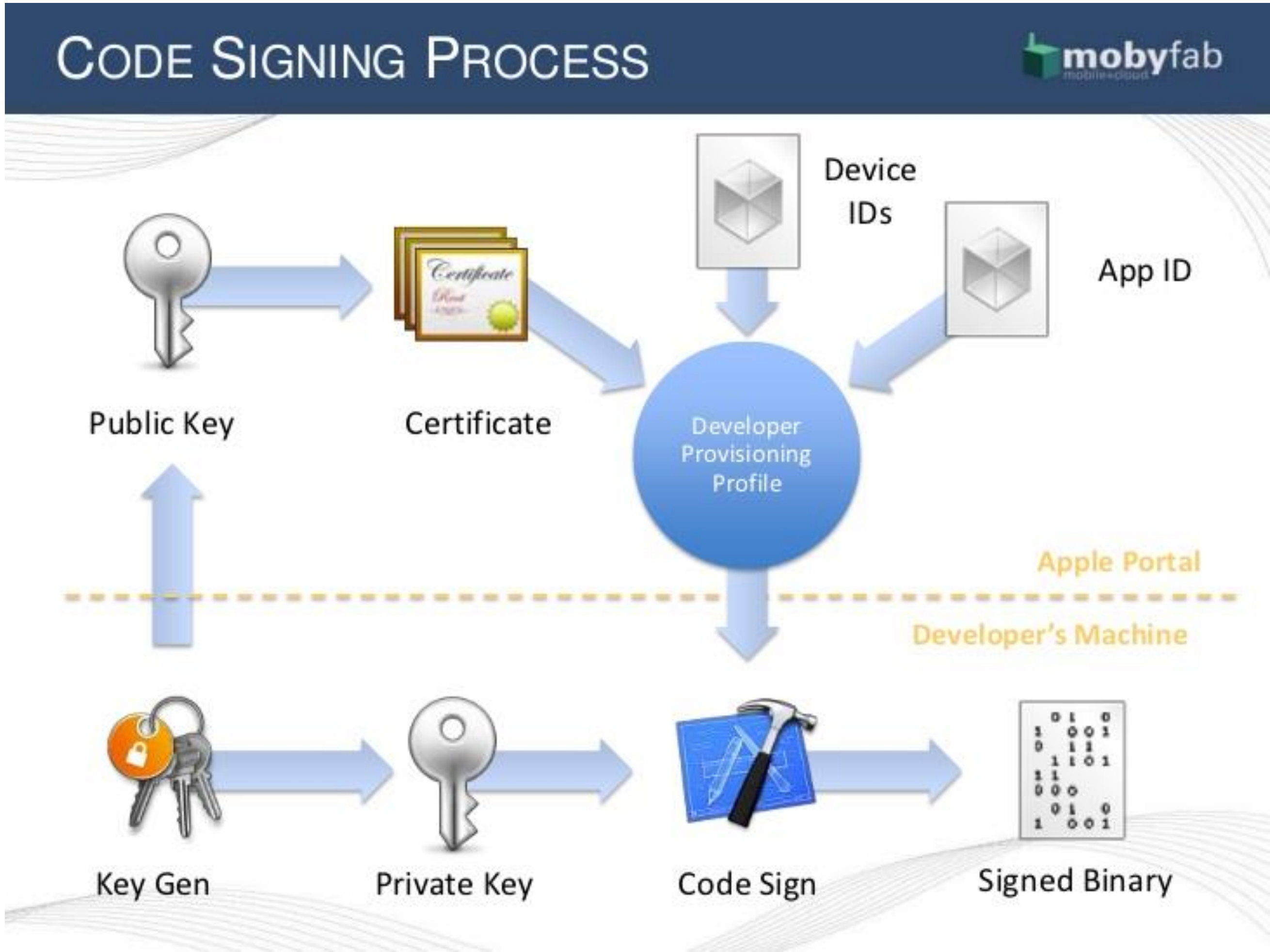
Code Signing: Developer

How to deploy apps as developer?

1. Generate private keys

2a. Certificate issued by Apple

2b. Specific certificates
→ <u>not trusted</u> on devices by default!



How to establish trust?

Using „Provisioning Profiles":

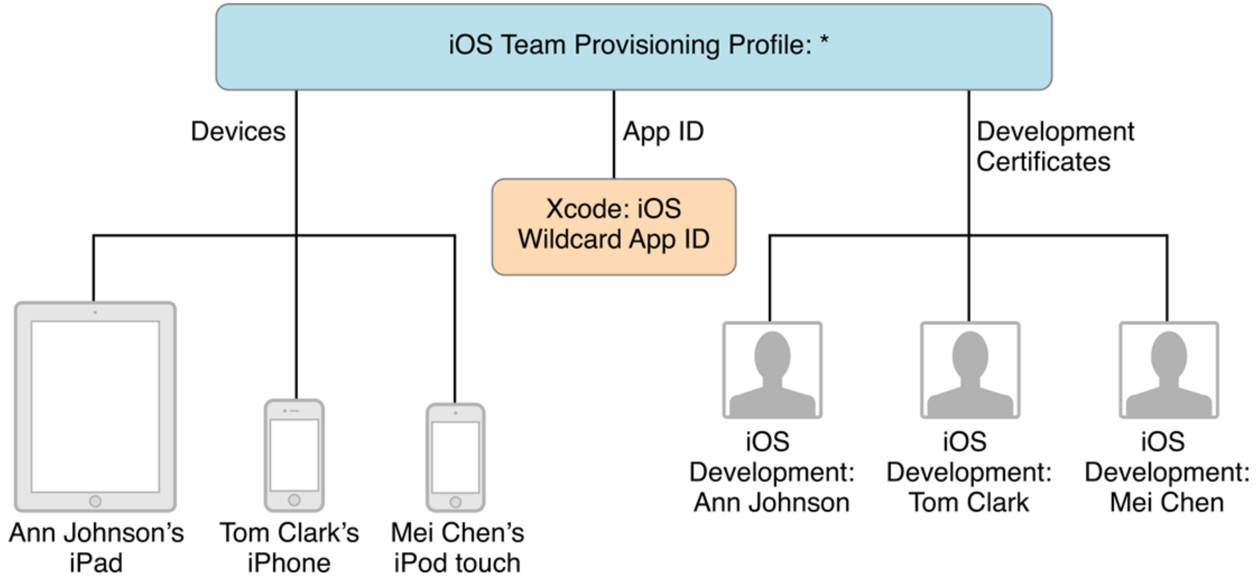
Set of iOS development certificates, unique device identifiers, and App ID

Code Signing: Enterprise

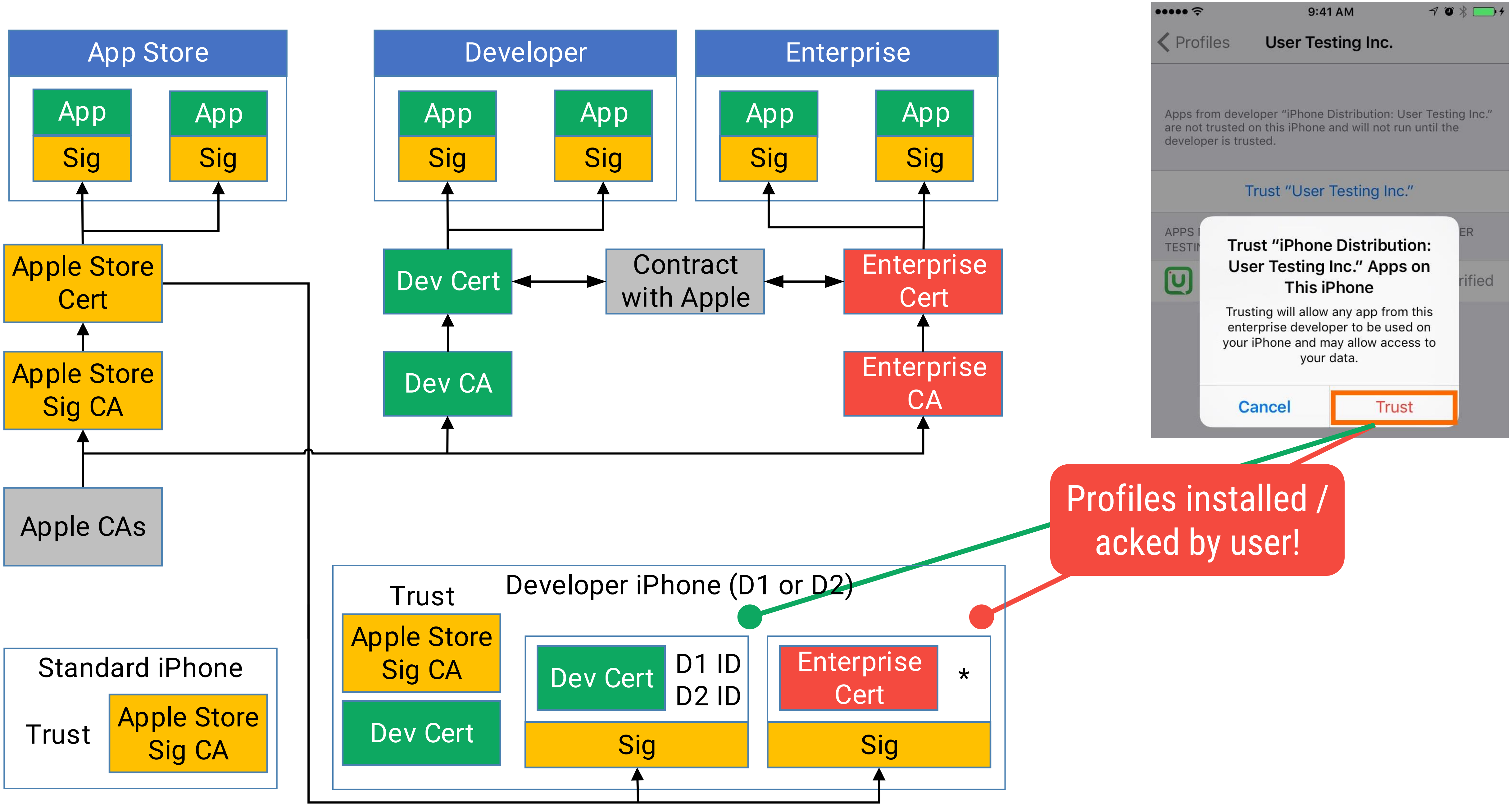
How to deploy apps as company?

- Like developer but multiple devices in „Team Provisioning Profile“
- Individually approved by Apple
- Companies can directly deploy anything (no AppStore submission!)
- User *implicitly* trusting all apps from same enterprise app store

  → Needed for MDM!



IAIK TU Graz



App Store
App
Sig
App
Sig
Developer
App
Sig
App
Sig
Enterprise
App
Sig
App
Sig
Apple Store Cert
Apple Store Sig CA
Apple CAs
Dev Cert
Contract with Apple
Enterprise Cert
Dev CA
Enterprise CA
9:41 AM
Profiles
User Testing Inc.
Apps from developer "iPhone Distribution: User Testing Inc." are not trusted on this iPhone and will not run until the developer is trusted.
Trust "User Testing Inc."
Trust "iPhone Distribution: User Testing Inc." Apps on This iPhone
Trusting will allow any app from this enterprise developer to be used on your iPhone and may allow access to your data.
Cancel
Trust
Profiles installed / acked by user!
Standard iPhone
Trust
Apple Store Sig CA
Developer iPhone (D1 or D2)
Trust
Apple Store Sig CA
Dev Cert
Dev Cert
D1 ID
D2 ID
Sig
Enterprise Cert
*
Sig

IAIK TU Graz

Sandbox

Interaction

1. App tells how it wants to interact
   – System grants (only) minimal rights to app
2. User action requires access to system APIs → granted transparently
   – Eg. open / save dialogs, drag & drop, paste

Protected access (only with entitlement)

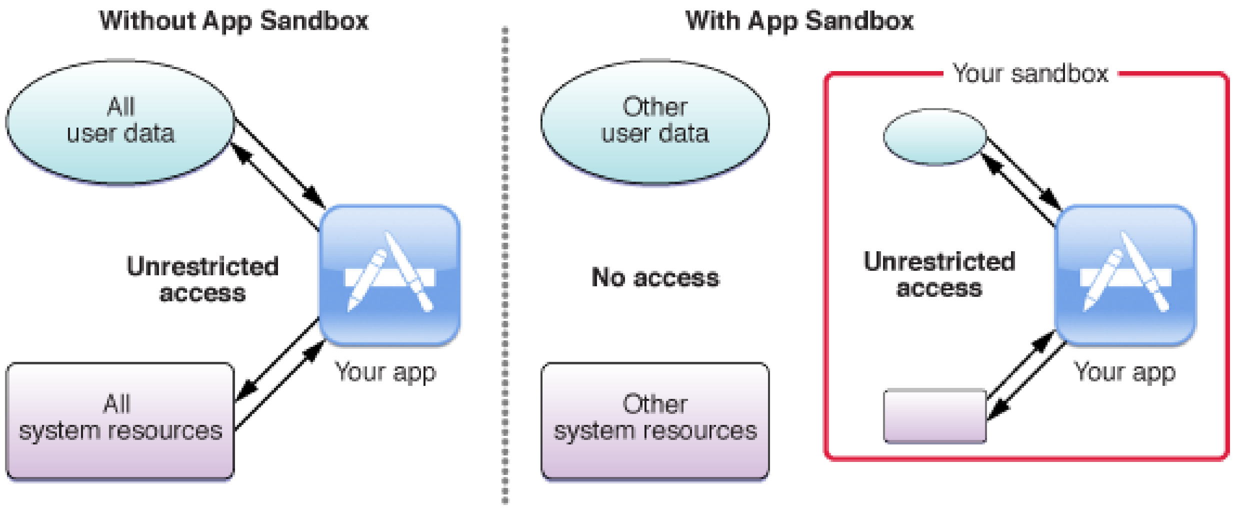
- Hardware (Camera, Microphone, …)
- Network Connections
- App Data (Calendar, Location, Contacts)
- User Files (Downloads, Music, Pictures, …)

Unprotected access (always possible): World-readable system files, invoke services

Sandbox

In Practice

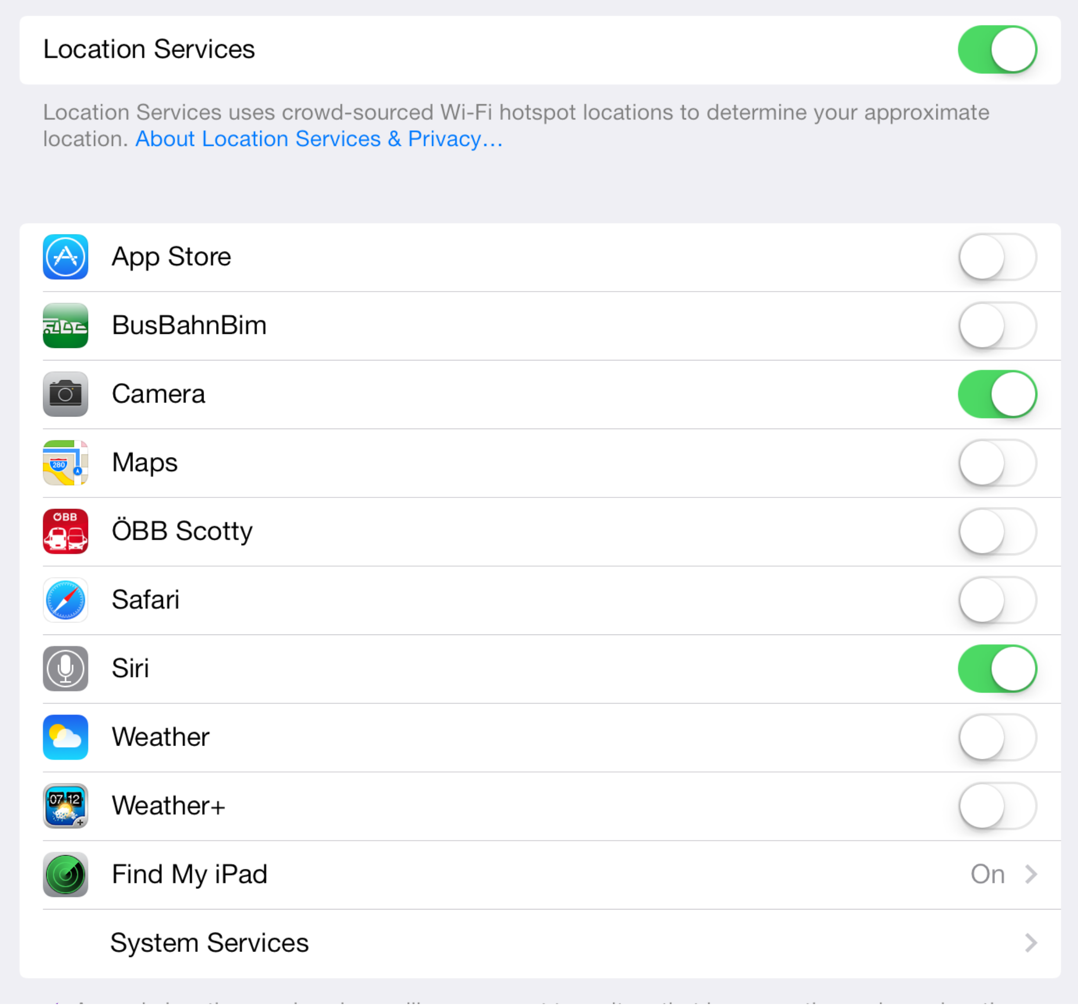
- Most apps run under same user *mobile*
  - Only few system apps & services as *root*
- Separate container for each app
  - Custom implementation of syscalls mmap and mprotect
    - Apps cannot set memory pages executable
    - Stop processes from executing dynamically generated code
  - App process restricted to own directory via chroot-like process
- Hardware driver access only via Apple frameworks



Source: https://goo.gl/SL4BCs

iOS Permissions

- No permission granting at installation
  - Only during runtime!
- Can be revoked in app settings
- Workflow
  - First API access: Request user
  - Further API access:
    Refer to saved permission state



Note: Only way to remove internet access for app
→ Turn off your WiFi / LTE connection…

iOS Permissions

- Apps do not *directly* request permissions
  - Developers do not have to specify which they want to use
  - Depending on use of sensitive APIs

- **Example:** App wants to access user‘s contacts
  - App calls method from `CNContactStore` class
  - Since iOS 10: Apps must present description how requested data is used
  - API access blocked until permission granted / denied

- ***Sensitive APIs***

Contacts, Microphone, Calendar, Camera, Reminders, Photos, Health, Motion Activity & Fitness, Speech Recognition, Location Services, Bluetooth Sharing, Media Library, Social Media Accounts

Malware?

- Reduced attack surface → stripped down OS
  - Lots of useful binaries missing, e.g. no `/bin/sh` → no „shell“ code ☹
  - Even if shell → no `ls`, `rm`, `ps`, etc.
  - With code execution, what could you do?

- Not many applications to attack
  - No Flash, Java
  - Mobile Safari does not render same files as desktop Safari (QT)

- Privilege separation
  - Most processes run as user „mobile“
    - Mobile Safari, Mobile Mail, Springboard, etc
  - Many resources require root privileges

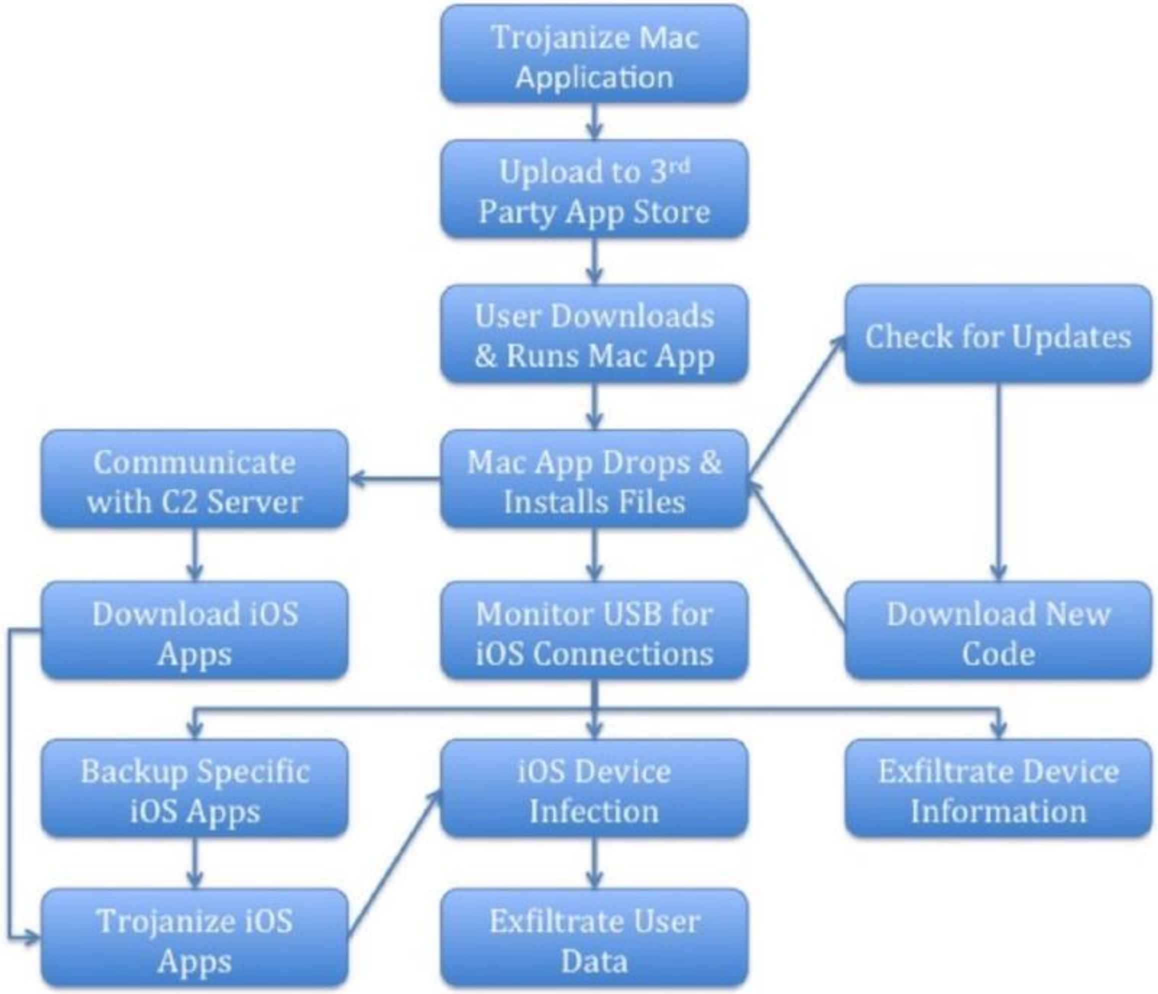
IAIK TU Graz

Wirelurker Malware

- Maiyadi App Store
  - 3rd Party Mac AppStore in China
  - Hosts „free“ apps
- Code signatures can be disabled on OS X

Attack scenario

1. OS X infection
2. App installed via USB on iPhone, signed with enterprise app store cert (User has to trust Provisioning profile!)
3. On normal (not profile trusting) phones: Not malicious but botnet contact



Source: https://goo.gl/tirnTD

Wirelurker Malware

Solution

Apple has to revoke enterprise certificate
→ If certificate revoked, apps cannot be started anymore

Detailed info: https://www.zdziarski.com/blog/?p=4140

Inferred problems

- Protect iTunes pairing better!
- Code Signature Certificate Pinning
- Accept enterprise provisioning profiles with one-click
  - Why are they needed for standard devices in the first place?

IAIK TU Graz

App Internals

App Types

From Apple

- Compiled into kernel, less restrictive
- Can: open SMS database but can not: send SMS, fork()
- Also run in sandbox: Mobile Safari, Mobile Mail, Mobile SMS
  - As user *mobile*

From App Store

- More restrictive sandbox
- Cannot access most of file system
  - Generally restricted to app's home directory
- Further restrictions on API usage by Apple
  - Data Protection for files and databases

App Files

- Distributed in **IPA format** (“iOS App Store Package”)
- ZIP archive with all code + resources

```
$ unzip SuperPassword.ipa –d acndemo
$ ls -R acndemo/

/Payload/SuperPassword.ipa/           App itself + static resources
       -> SuperPassword               “Fat Binary” executable (ARM-compiled code)
       -> Info.plist                  Bundle ID, version number, app name to display
       -> MainWindow.nib              Default interface to load when app is started
       -> Settings.bundle             App-specific preferences for system settings
       -> further resources           Language files, images, sounds, more GUI layouts (nib)
/iTunesArtwork                        512x512 pixel PNG image -> app icon
/iTunesMetadata.plist                 Developer name + ID, bundle identifier,
                                      copyright information, etc.
```

App Installation

- Until iOS 8
  - Unpacking to /var/mobile/applications/<APP_UUID>
  - APP_UUID = 128-bit number to uniquely identify app

- Since iOS 10
  - /private/var/mobile/Containers/Bundle/Application/<APP_UUID>/
    - App bundle (ARM binary, static resources)
    - Content of this folder used to validate code signature of app

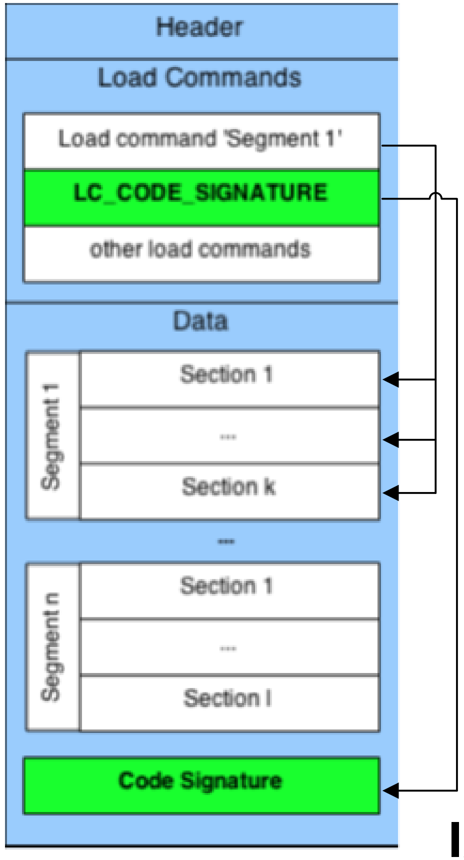
  - /private/var/mobile/Containers/Data/Application/<APP_UUID>/
    - User-generated app data
    - Subfolder „Library“: Cookies, caches, preferences, configuration files (plist)
    - Subfolder „tmp“: Temp files for current app launch only (not persisted)

  - /private/var/mobile/Containers/Shared/AppGroup/<APP_UUID>/
    - To share with other apps & extensions of same app group

iOS Executable

- „Fat Binary“ → Includes bins for ARMv7, ARMv8, …

- Each bin is in Mach-O format
  - Header
    - Identification
    - Architecture
  - Load commands
    - Virtual Memory Layout
    - Libraries
    - Code signature
    - Encryption
  - Data
    - Executable code
    - Read / write data
    - Objective C runtime information



IAIK TU Graz

iOS App Analysis

IAIK TU Graz

Application Analysis

→ Traditionally two approaches

- Dynamic Analysis: Monitor live file access using jailbroken device
- Static Analysis: Look for file API calls + parameters in binary dump

Challenge?

- iOS apps are compiled down to native code
  - Analysis on disassembly, e.g. using Hopper or IDApro
  - Hard to find the needle in the haystack

- How do you get apps for analysis?
  - All binaries encrypted by Apple → decryptable but anyway...
  - Need jailbroken device but jailbreaking is no „feature by design“

Case Study: Viber

Encryption appears to be custom C++ implementation

```
-[VIBEncryptionContext initWithContext:]
-[VIBEncryptionContext context]
-[VIBEncryptionContext params]
-[VIBEncryptionContext setParams:]
-[VIBEncryptionContext .cxx_destruct]
-[VIBEncryptionManager initWithInjector:]
-[VIBEncryptionManager dealloc]
-[VIBEncryptionManager checkEncryptionAbilityForAttachment:completion:]
-[VIBEncryptionManager checkEncryptionForConversation:completion:]
-[VIBEncryptionManager beginEncryptionWithContext:]
-[VIBEncryptionManager encryptData:length:withContext:]
-[VIBEncryptionManager endEncryptionWithContext:]
-[VIBEncryptionManager popEncryptionParamsForContext:]
-[VIBEncryptionManager encryptData:encryptionKey:]
-[VIBEncryptionManager calculateMD5ForAttachment:]
-[VIBEncryptionManager decryptAttachment:completion:]
-[VIBEncryptionManager decryptData:withEncryptionParams:]
-[VIBEncryptionManager decryptFile:withEncryptionParams:]
-[VIBEncryptionManager handleSecureStateChanged:]
-[VIBEncryptionManager supportedMediaTypes]
-[VIBEncryptionManager .cxx_destruct]
```

Case Study: Viber

```
000632fa    str     r4, [sp, #0x100 + var_100]
000632fc    movw    r2, #0x412e                 ; @"Viber can not verify this number. This may be the result of an error or a breach.\\nPlease verify %@ agai
00063300    movt    r2, #0xd9                   ; @"Viber can not verify this number. This may be the result of an error or a breach.\\nPlease verify %@ agai
00063304    mov     r1, r6                      ; argument #2 for method imp___picsymbolstub4__objc_msgSend
00063306    add     r2, pc                      ; @"Viber can not verify this number. This may be the result of an error or a breach.\\nPlease verify %@ agai
00063308    mov     r3, r8
0006330a    mov     r5, r0
0006330c    blx     imp___picsymbolstub4__objc_msgSend
00063310    mov     r7, r7
00063312    blx     imp___picsymbolstub4__objc_retainAutoreleasedReturnValue
00063316    str     r0, [sp, #0x100 + var_C8]
00063318    mov     r0, r5
0006331a    blx     imp___picsymbolstub4__objc_release
0006331e    ldr.w   r0, [fp]                    ; objc_cls_ref_NSBundle,_OBJC_CLASS_$_NSBundle, argument #1 for method imp___picsymbolstub4__objc_msgSend
00063322    mov     r1, sl
00063324    blx     imp___picsymbolstub4__objc_msgSend
00063328    mov     r7, r7
0006332a    blx     imp___picsymbolstub4__objc_retainAutoreleasedReturnValue
0006332e    str     r4, [sp, #0x100 + var_100]
00063330    movw    r2, #0x410a                 ; @"Messages sent by participants in this conversation are encrypted and %@ is Verified", :lower16:(cfstring_
00063334    movt    r2, #0xd9                   ; @"Messages sent by participants in this conversation are encrypted and %@ is Verified", :upper16:(cfstring_
00063338    mov     r1, r6                      ; argument #2 for method imp___picsymbolstub4__objc_msgSend
0006333a    add     r2, pc                      ; @"Messages sent by participants in this conversation are encrypted and %@ is Verified"
0006333c    mov     r3, r8
0006333e    mov     r5, r0
00063340    blx     imp___picsymbolstub4__objc_msgSend
00063344    mov     r7, r7
00063346    blx     imp___picsymbolstub4__objc_retainAutoreleasedReturnValue
0006334a    str     r0, [sp, #0x100 + var_B8]
0006334c    mov     r0, r5
0006334e    blx     imp___picsymbolstub4__objc_release
00063352    ldr.w   r0, [fp]                    ; objc_cls_ref_NSBundle,_OBJC_CLASS_$_NSBundle, argument #1 for method imp___picsymbolstub4__objc_msgSend
00063356    mov     r1, sl
00063358    blx     imp___picsymbolstub4__objc_msgSend
0006335c    mov     r7, r7
0006335e    blx     imp___picsymbolstub4__objc_retainAutoreleasedReturnValue
00063362    str     r4, [sp, #0x100 + var_100]
00063364    movw    r2, #0x40e6                 ; @"This conversation cannot be encrypted. This may be the result of an error or a geo-location limitation",
00063368    movt    r2, #0xd9                   ; @"This conversation cannot be encrypted. This may be the result of an error or a geo-location limitation",
0006336c    mov     r1, r6                      ; argument #2 for method imp___picsymbolstub4__objc_msgSend
0006336e    add     r2, pc                      ; @"This conversation cannot be encrypted. This may be the result of an error or a geo-location limitation"
00063370    mov     r3, r8
00063372    mov     r5, r0
```

Case Study: WhatsApp

```
$ cd /private/var/mobile/Containers/Shared/AppGroup
$ ls -l 332A098D-368C-4378-A503-91BF33284D4B/

-> Axolotl.sqlite
-> ChatSearch.sqlite
-> ChatStorage.sqlite
-> Contacts.sqlite
-> StatusList.plist
-> SyncHistory.plist
-> calls.backup.log
...
```

- Deleting messages from WhatsApp → message still in SQLite DB
  - Deleting SQLite records sets them free but does not clear them
  - Can be recovered as long as not overwritten

See: https://goo.gl/nce4jo

IAIK TU Graz

Case Study: WhatsApp

```
$ sqlite3 ChatStorage.sqlite
SQLite version 3.8.4.3 2014-04-03 16:53:12
Enter ".help" for usage hints.

sqlite> .tables
ZWABLACKLISTITEM  ZWAGROUPINFO     ZWAMESSAGE    Z_METADATA      ZWACHATPROPERTIES
ZWAGROUPMEMBER    ZWAMESSAGEINFO   Z_PRIMARYKEY  ZWACHATSESSION  ZWAMEDIAITEM
ZWAMESSAGEWORD
```

- Messages - ZWAMESSAGE
  - Also in file ChatSearch.sqlite
- Open chats - ZWACHATSESSION
  - Single user & group chats
- Media location - ZWAMEDIAITEM
- ...

See: https://goo.gl/bfXqG

Database view | File Info

ZMESSAGEDATE	ZSENTDATE	ZFROMJID	ZMEDIASECTIONID	ZPHASH	ZPUSHNAME	ZSTANZAID	ZTEXT
438344687		27-14166 51887@g.us				9BFCF037952062F08F	
438344687		27-14166 51887@g.us				8B30B691B63744F4A3	
426673193		22-14049 80393@g.us				81763AB90957B4E460	
426673193		22-14049 80393@g.us				0D02DB95AE3230C30A	
483635628,093624		27-14166 51887@g.us				3EA65954161BBF4605	
483637174,381004		18@s.wha tsapp.net				46928ABCAD52AAD45C	
483637173,891472	483637174					DF91BE5FC5C7DE68C9	Ehiiii
483641447		18@s.wha tsapp.net	2016-04		Beaa	4326AEE22C1BFF3146	
483644648		83@s.wha			Jack	A58883CBCB8877791B4	Eii

Case Study: Telegram

- Lots of data also stored in Shared directory
- Documents folder contains tgdata.db
  - Contains all information about contacts, conversations, files exchanged, etc.
  - SQLite db → recovery of deleted chats possible as with WhatsApp
  - Tables
    - messages_v29: List of all exchanged messages
    - conversations_v29: List of active chats
    - encrypted_cids_v29: Conversation IDs of secret chats

```
sqlite> SELECT * FROM encrypted_cids_29;

encrypted_id = 1824030108
         cid = -2147483648

encrypted_id = ...
         cid = ...
```

```
sqlite> SELECT * FROM messages_v29;

cid = -2147483648
message = Once I was a secret chat...
from_id = 243610671
to_id = -2147483648
...
```

Case Study:
Crypto Misuse in iOS Applications

Paper: Automated Binary Analysis on iOS - A Case Study on Cryptographic Misuse in iOS Applications. Feichtner, J., Missmann, D. & Spreitzer, R. 2018 Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks. New York: ACM, New York, p. 236-247 12 p.

IAIK TU Graz

Challenges

- Decompiling machine code
  - No(?) ARMv8 64-bit decompiler to LLVM IR available
- Language pecularities
  - Dynamic control-flow decisions during runtime → information flow?
  - Information about types lost during compilation (but still in binary!)
- Pointer analysis
  - Where do different variables point to during execution?
  - How to deal with aliasing?
  - Potential trade-off: accuracy of slides <-> runtime overhead of points-to analysis

Our Solution

- Framework to automatically track *definable* method invocations in iOS apps
- General design but study focus on misconceptions in crypto API usage

Features

- Generic decompiler for ARMv8 64-bit → LLVM IR code
  - Also handles language pecularities of iOS binaries
- Pointer Analysis
  - Handle Aliasing, reconstruct original call graph
- Static Slicing
  - Extract individual execution paths for parameter backtracking
- Evaluates „security rules“

Source Code: https://github.com/IAIK/ios-analysis

Security Rules

1. No ECB mode for encryption
2. No non-random IV for CBC encryption
3. No constant encryption keys
4. No constant passwords or salts for PBE
5. Not fewer than 1000 iterations for PBE
6. Do not use static seeds to seed SecureRandom

Proposed by Egele et al.: CryptoLint

Goals

- Transform these “common sense” rules for iOS
  - Different defaults (CBC instead of ECB), Rule 6 cannot be violated on iOS
  - Adapted for system crypto provider *CommonCrypto*
- Automatically check these issues in arbitrary apps

IAIK TU Graz

„No non-random IV for CBC encryption“

Problem

- IV constant or predictable → deterministic / stateless encryption scheme
- Susceptible to *Chosen-Plaintext Attack*

Our „Security Rule“

- Precondition: Cipher uses CBC mode

```
CCCryptorStatus CCCryptorCreate(
    CCOperation op,             /* kCCEncrypt, etc. */
    CCAlgorithm alg,            /* kCCAlgorithmDES, etc. */
    CCOptions options,          /* kCCOptionPKCS7Padding, etc. */
    const void *key,            /* raw key material */
    size_t keyLength,
    const void *iv,             /* optional initialization vector */
    CCCryptorRef *cryptorRef);  /* RETURNED */
```

- Slicing criteria

```
CCrypt(...,X5,...), CCCryptorCreate(...,X5,...), CCCryptorCreateWithMode(...,X4,...)
```

- IV should be “random” / generated by cryptographically secure RNG, e.g. using
  - CCRandomGenerateBytes() in *CommonCrypto* or
  - SecRandomCopyBytes() in *Security* library

Evaluation Scenario

Motivation

- *„Does our framework also perform with real-world applications?“*
- *„What are our security rules able to cover?“*
- *„Do iOS developers know how to apply crypto APIs correctly?“ :-)*

Method & Dataset

- Manual analysis
  - 15 open-source apps from Github using *CommonCrypto*
    - Refined framework / security rules where necessary
    - Validated execution paths manually using source codes
- Automated analysis
  - 634 free applications from official iOS App Store (> 10.000 installations each)
  - Only apps where crypto usage seemed obvious, e.g. password managers

Evaluation Results

Framework

	Count	[%]
Downloaded from iOS App Store	634	
No *CommonCrypto* calls	139	22%
With *CommonCrypto* calls	495	78%
Binary only for ARMv7	7	1%
Not decompilable	46	9%
Out of memory	25	5%
Analyzable with *CommonCrypto* calls	417	84%

Security rules

Violated Rule	# Applications	[%]
Rule 2: Uses non-random IV	289	69%
Rule 3: Uses constant encryption key	268	64%
Rule 1: Uses ECB mode	112	27%
Rule 4: Uses constant salts for PBE	72	17%
Rule 5: Uses $<$ 1,000 iterations (PBE)	49	12%
Applications with $\geq$ 1 rule violations	343	82%
No rule violation	74	18%

Origin of constant secrets

	# Violations
Constant string used as encryption key	193
Constant password for PBKDF2	84
Hash value of constant string	18
Secret retrieved from *NSUserDefaults*	14
Constant key data	6
Applications violating rule 3	268

IAIK TU Graz

Limitations

Framework

- Context- and field-insensitive approach
  - Parameter backtracking might also track spurious execution paths
- UI elements
  - E.g. backtracking password input might end at externally defined *UITextField* object

Security Rules

- Not aware of custom implementations / 3rd party crypto libs

- Only evaluate what you specify…
  - „Home-brew“ encryption keys fly below the radar…
  - Passwords padded with NULL bytes / truncated to key length count as „non-constant“ input

IAIK TU Graz

Conclusion

- Novel approach to tackle automated analysis of iOS applications
  - ARMv8 64-bit decompiler
  - Pointer Analysis
  - Static Slicing
  - Parameter Backtracking

- Case Study on 417 applications using crypto APIs
  - Security rules targeting common crypto misuse
  - Iteratively refined approach using open-source applications

→ *343 / 417 (82%) apps violate at least one security rule*
Mostly: Use of non-random IV (69%), constant keys (64%), ECB mode (27%)

IAIK TU Graz

Outlook

- 30.04.2020
  - Android Platform Security
- 07.05.2020
  - Application Security on Android

IAIK TU Graz