State of iOS Jailbreaking in 2025

Lars Fröder

Cellebrite Labs

About Me

- Security Researcher from Germany
- Started iOS development journey in 2017, research in 2022
- Employed at Cellebrite Labs

- Developed various iOS jailbreak system extensions (“tweaks”)
- Developer of TrollStore and Dopamine Jailbreak

@opa334.bsky.social

opa334@infosec.exchange

Motivations

- Run unsigned / third party software on iPads and iPhones
- Enable system introspection capabilities (e.g. Frida, lldb)
- Load system extensions / tweaks

Agenda

- Code Signing on iOS
- TrollStore 
- Dopamine 

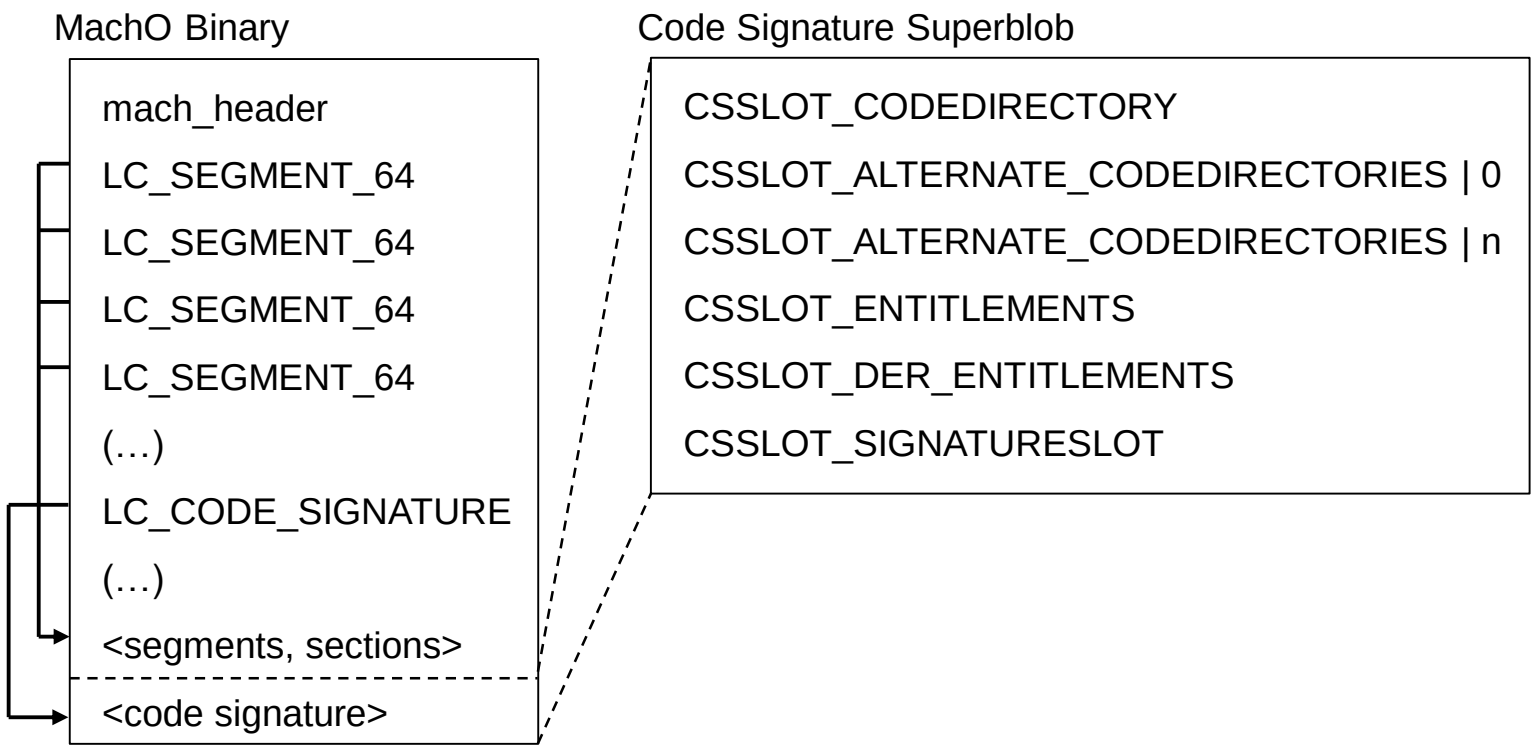
Agenda

- Code Signing on iOS
- TrollStore 
- Dopamine 

Code Signing on iOS

- Used for Apple to maintain control over all authorized software
-
- Main thing that a jailbreak needs to bypass
- Enforces all executables are either
  - Shipped with the operating system (Ad-hoc signed)
  - Distributed on the App Store (Apple signed)
  - Installed via a Developer account (Developer signed)

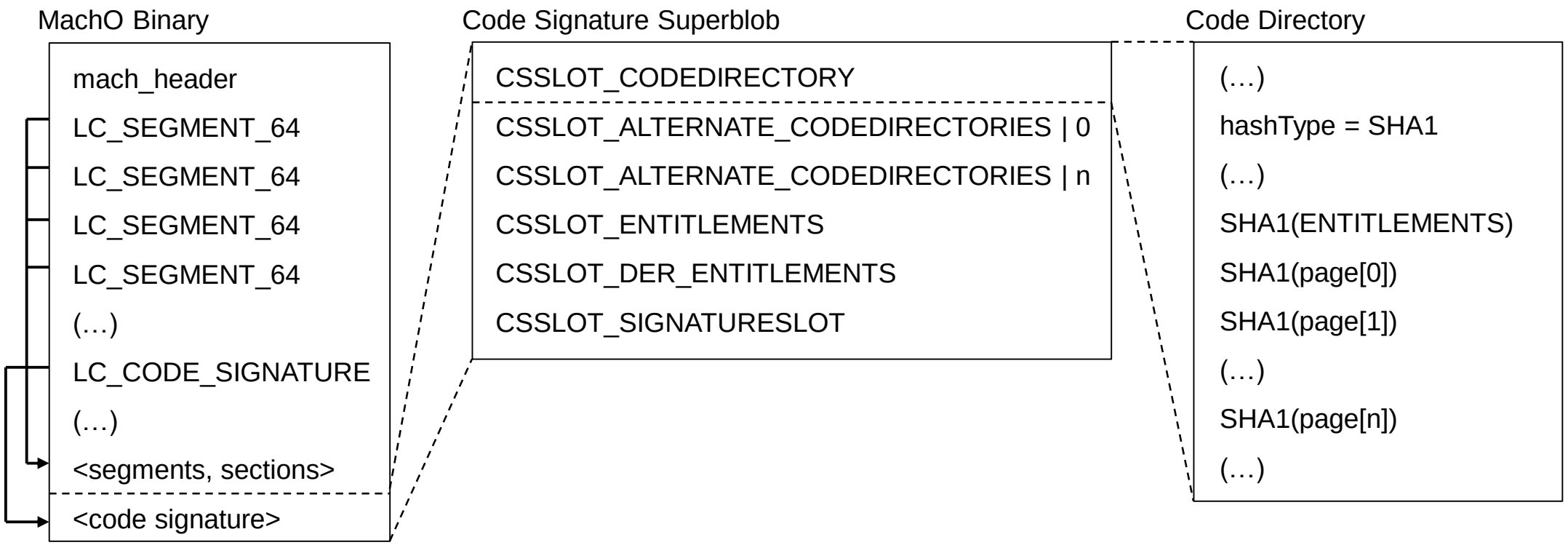
Code Signature



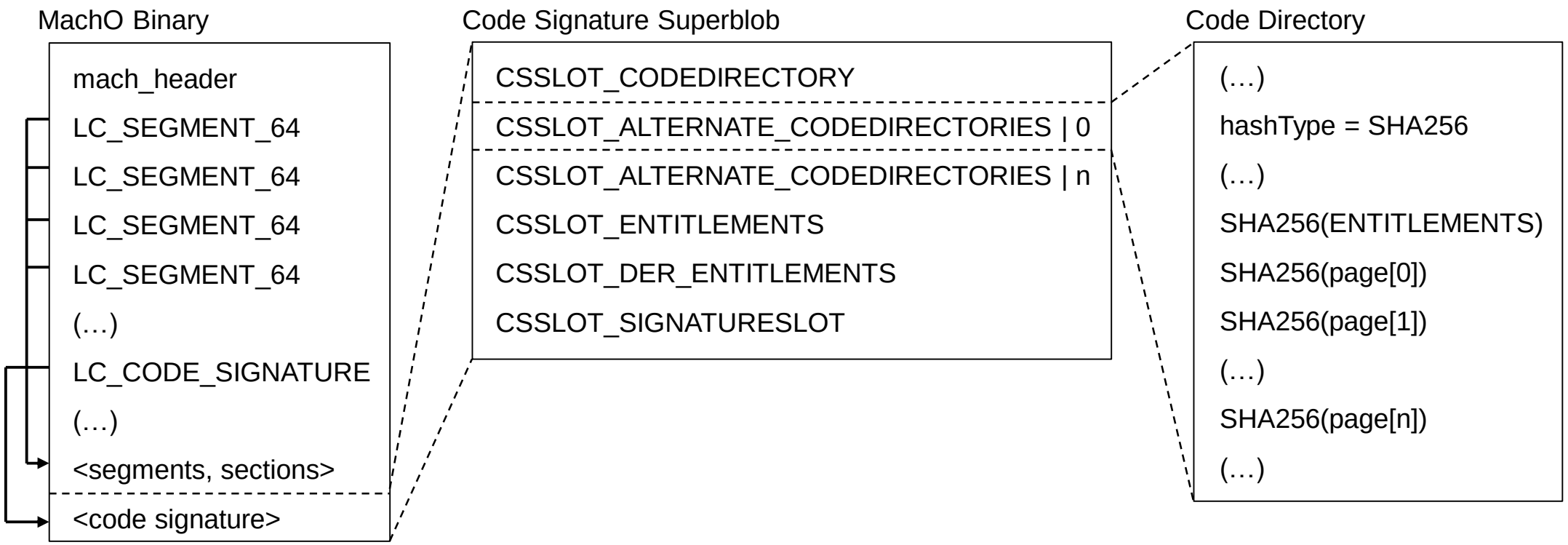
Code Signature: Code Directory

- Contain hashes of all executable pages within the binary
- Additionally contain hashes of other relevant data (e.g. CSSLOT_ENTITLEMENTS)
- One code directory has one hash type, (e.g. SHA1, SHA256, …)
- One binary can have multiple code directories with different hash types

Code Signature: Code Directory



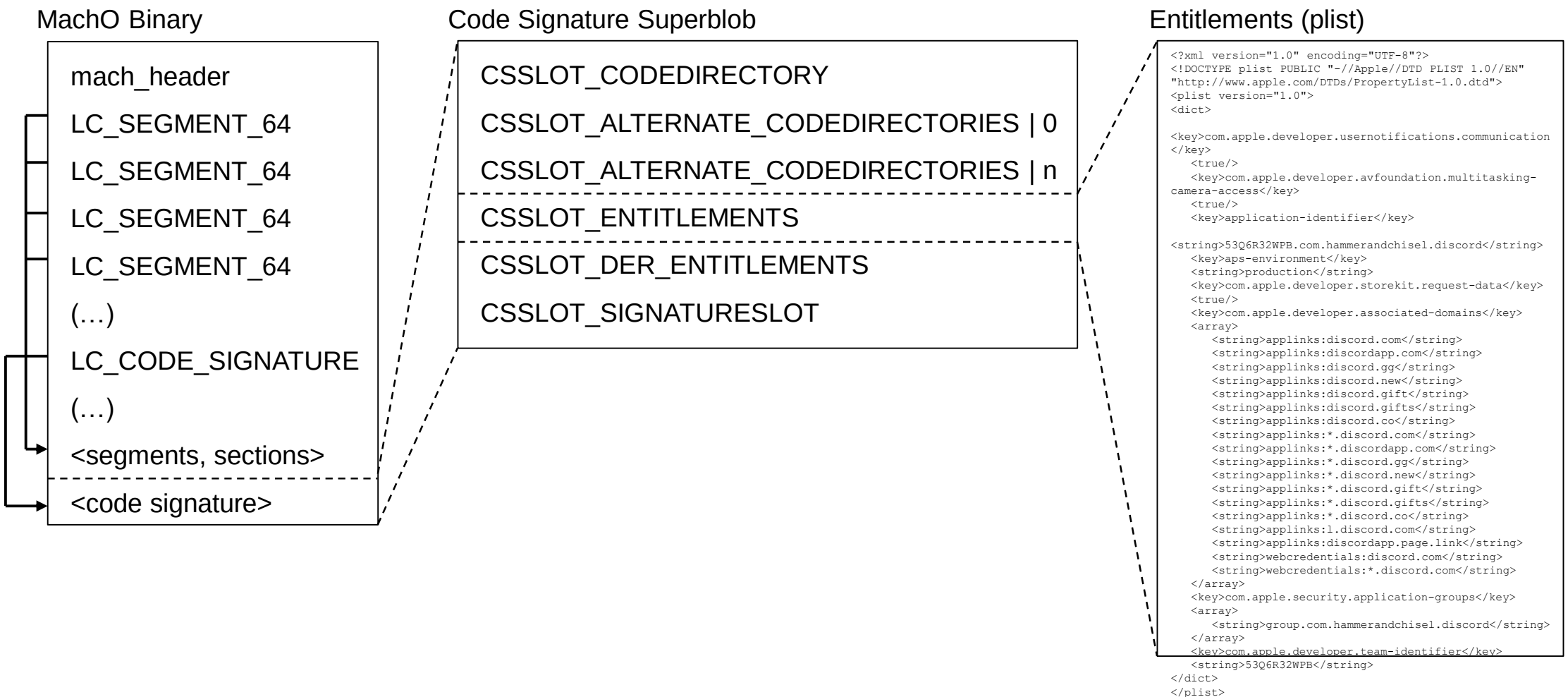
Code Signature: Code Directory



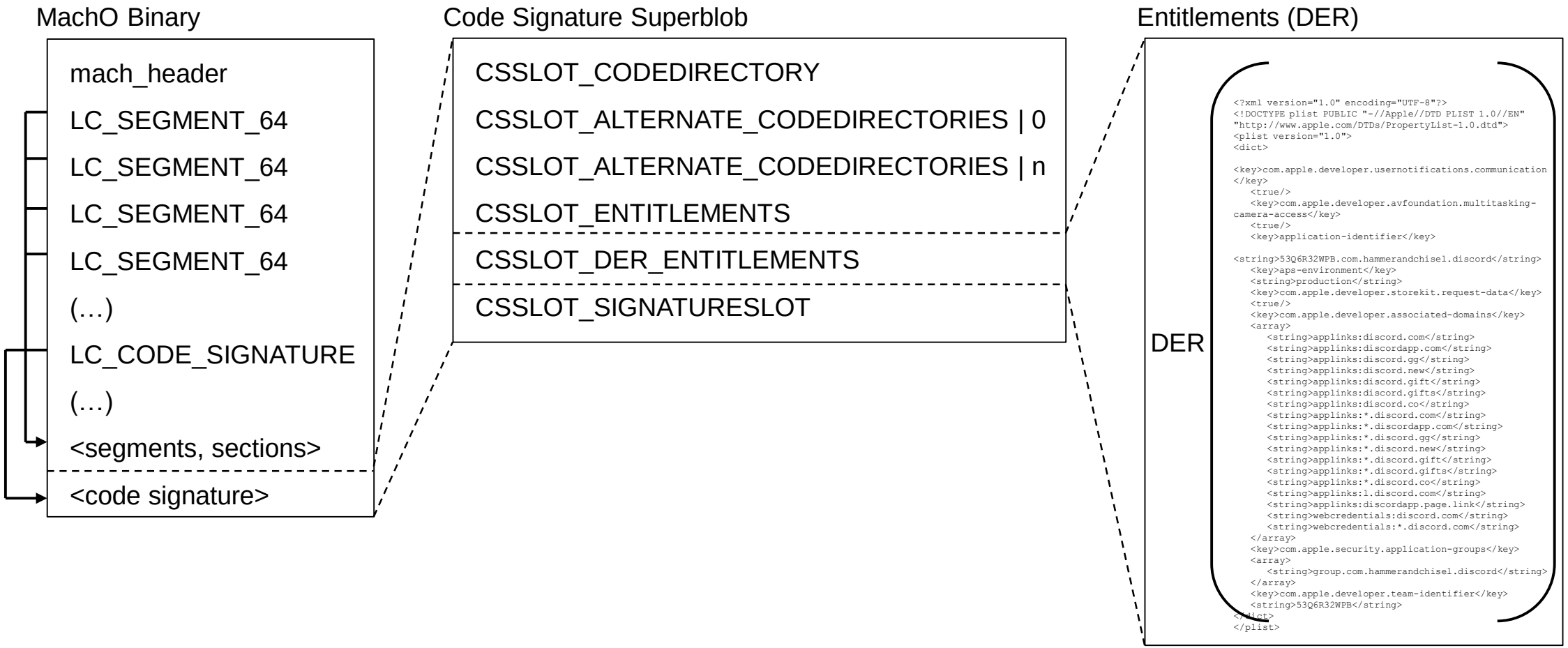
Code Signature: Entitlements

- Describe the permissions of the binary
  - Kernel drivers it can access
  - File paths it can read/write to/from
  - Whether the binary is sandboxed
  - Whether the binary may be debugged by other processes
  - etc…
- Can be checked both by the Kernel itself and by other processes

Code Signature: Entitlements



Code Signature: Entitlements

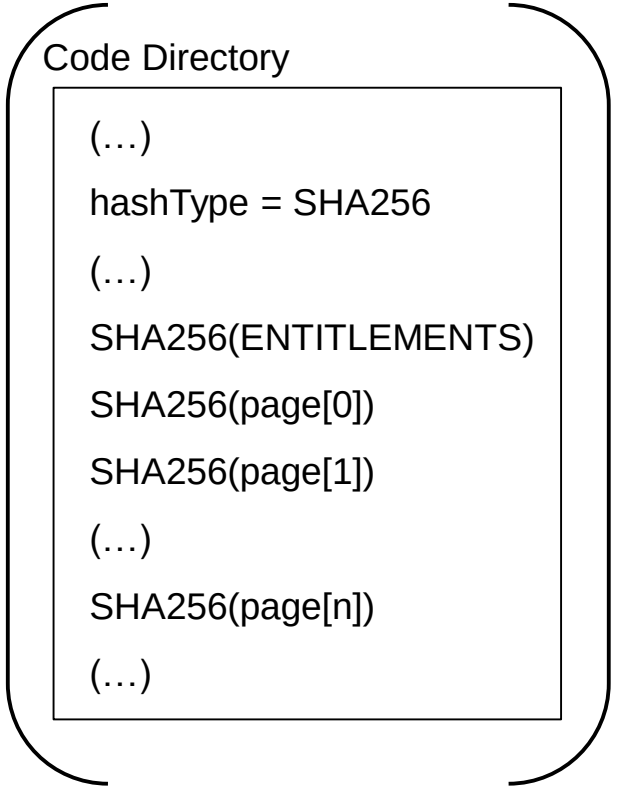


Code Signature: Signature Blob

- Contains cryptographically signed hash of code directories
- Signed with Apple or Developer cert
- Adhoc signed binaries do not have a signature, those are verified via the TrustCache
- Can be signed by multiple signers

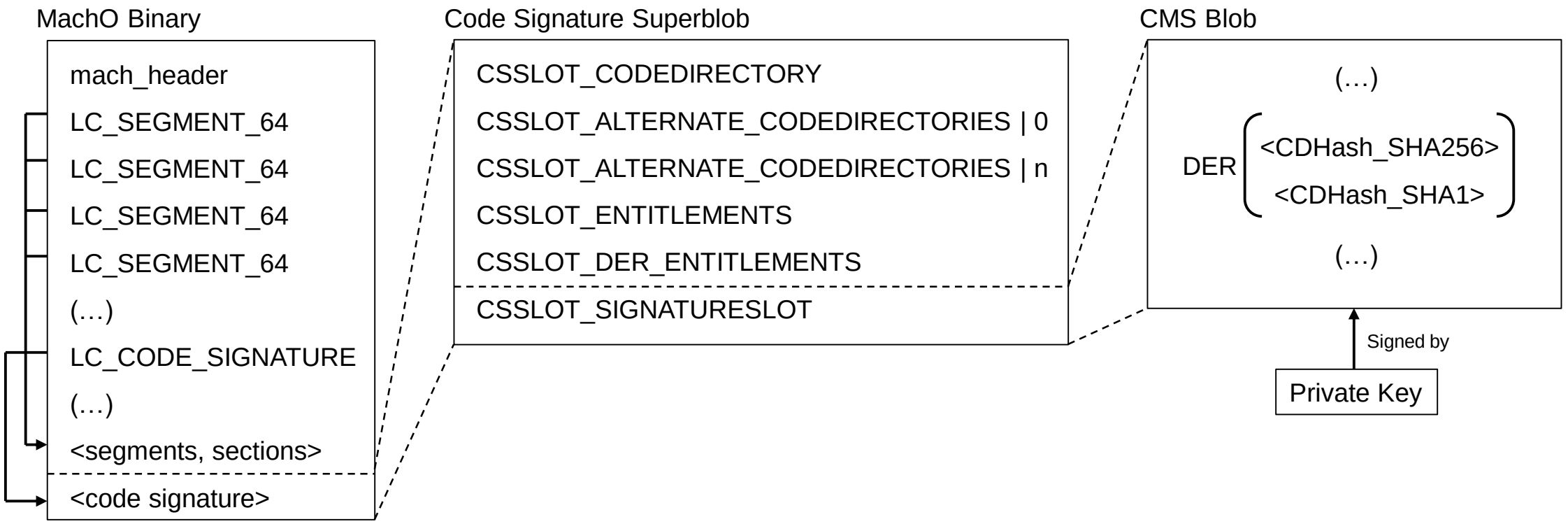
Code Signature: Code Directory Hash

CDHash_SHA256 = SHA256



- Uniquely identifies executable / library
- Hash of all other hashes and metadata -> Ensures integrity
- Contained within Signature Blob, recalculated and compared when validating file

Code Signature: Signature Blob



Code Signature Enforcement

- Every executables requires a valid code signature to run
  - System executables: Ad hoc signed, verified by CDHash in trust cache
  - AppStore executables: Signed by Apple on submission using “Apple iPhone OS Application Signing” certificate
  - Xcode App executables: Signed by Developer certificate issued by Apple, extremely limited
- Checked by the Kernel on execution
  - During posix_spawn or execve syscall

Code Signing Enforcement: Trust Level

- Trust Levels are used for isolation between different process “types”

- A process cannot dlopen / mmap a library with a lower trust level than the executable that the process was spawned from
- A process cannot obtain a task port (e.g. debugging rights) for a process with a higher trust level than the executable that the caller process was spawned from

Code Enforcement Paths: System Binaries

- Kernel checks whether CDHash of binary is considered trustworthy, this is true if
  - The CDHash is contained within a static list of CDHashes shipped with the operating system
  - The CDHash is contained within one of multiple dynamic lists of CDHashes that can be loaded by Xcode at runtime
- If it finds a match, the process is spawned with a trust level of 8

Code Enforcement Paths: App Store Binaries

- Kernel calls into Apple Mobile File Integrity driver
- Apple Mobile File Integrity calls into CoreTrust driver
- CoreTrust parses signature and ensures the binary is validly signed by App Store certificate (public key embedded into operating system)
- If it is, the process is spawned with a trust level of 7

Code Enforcement Paths: Developer Signed

- Only allowed when developer mode is enabled
- If App Store check fails, CoreTrust contacts the userspace service amfid to verify whether a valid developer certificate signs the binary
- If this check is successful, the process is spawned with a trust level of 5

Code Enforcement Paths

Trust Level	Type	Checked by
8	In TrustCache	Kernel (CSM / PMAP_CS / TXM)
7	App Store	CoreTrust
5	Developer Signed	amfid

(Simplified for better accessibility)

Agenda

- Code Signing on iOS
- TrollStore 
- Dopamine 

TrollStore (iOS 14.0 - 16.6.1, 17.0)

CVE-2023-41991

Security

Available for: iPhone XS and later, iPad Pro 12.9-inch 2nd generation and later, iPad Pro 10.5-inch, iPad Pro 11-inch 1st generation and later, iPad Air 3rd generation and later, iPad 6th generation and later, iPad mini 5th generation and later

Impact: A malicious app may be able to bypass signature validation. Apple is aware of a report that this issue may have been actively exploited against versions of iOS before iOS 16.7.

Description: A certificate validation issue was addressed.

CVE-2023-41991: Bill Marczak of The Citizen Lab at The University of Toronto's Munk School and Maddie Stone of Google's Threat Analysis Group

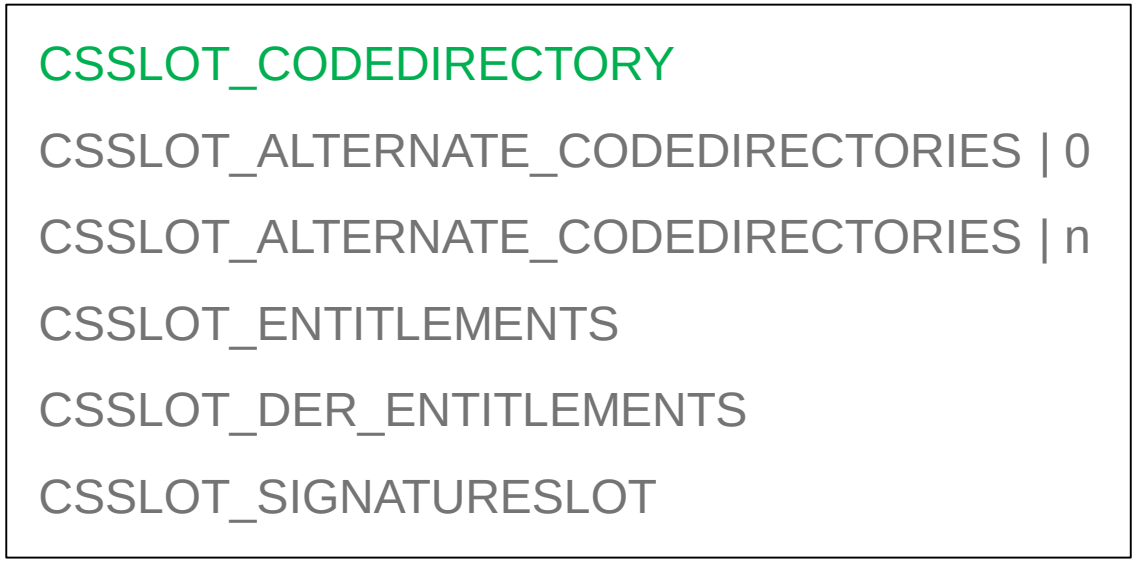
- Patched in iOS 17.0.1 and 16.7
- Reconstructed through patchdiffing by @alfiecg_dev and me

CVE-2023-41991

- Within the CoreTrust Kernel Extension, which is responsible for checking the code signature of App Store apps
- Called by AMFI (Apple Mobile File Integrity) Kernel Extension
- One of the most complex bugs I have ever seen
- Multiple quirks combined allow a fakesigned binary to run
  - CoreTrust only checks whether the last signer of a signature is valid (or rather, uses the same error variable for every signer, meaning it will just be overwritten by the last check)
  - CoreTrust returns the CodeDirectory hash of the first signer back to AMFI
  - CoreTrust only passes the first signer to the function that checks whether the binary is App Store signed

CVE-2023-41991: Code Directory

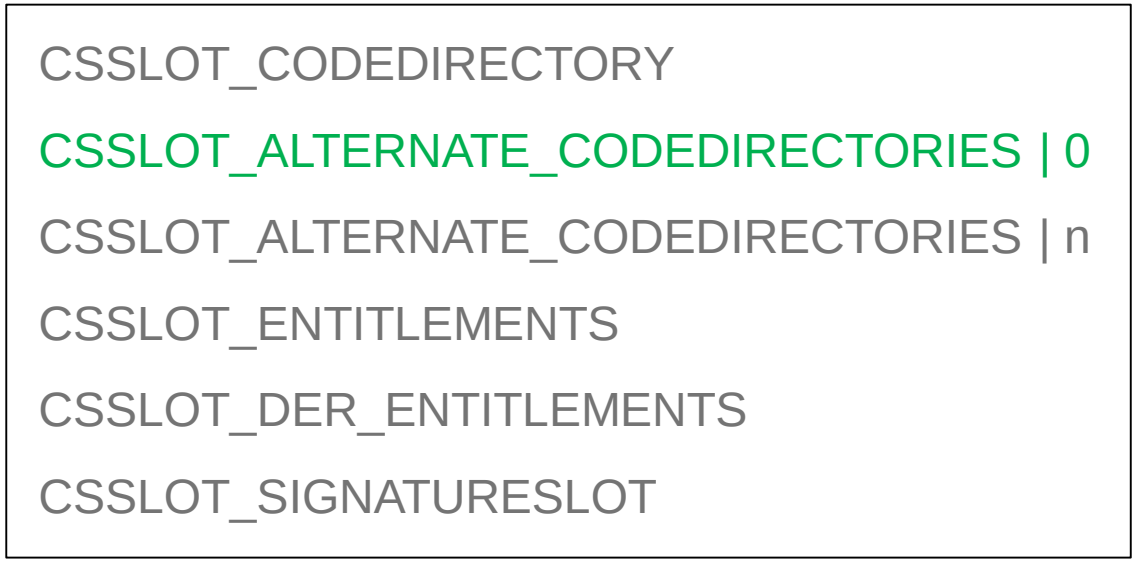
Code Signature Superblob



- Code Directory stolen from a validly signed App Store app
- Type: SHA1
- None of contained hashes match our binaries executable pages, nor our entitlements or anything else

CVE-2023-41991: Alternate Code Directory

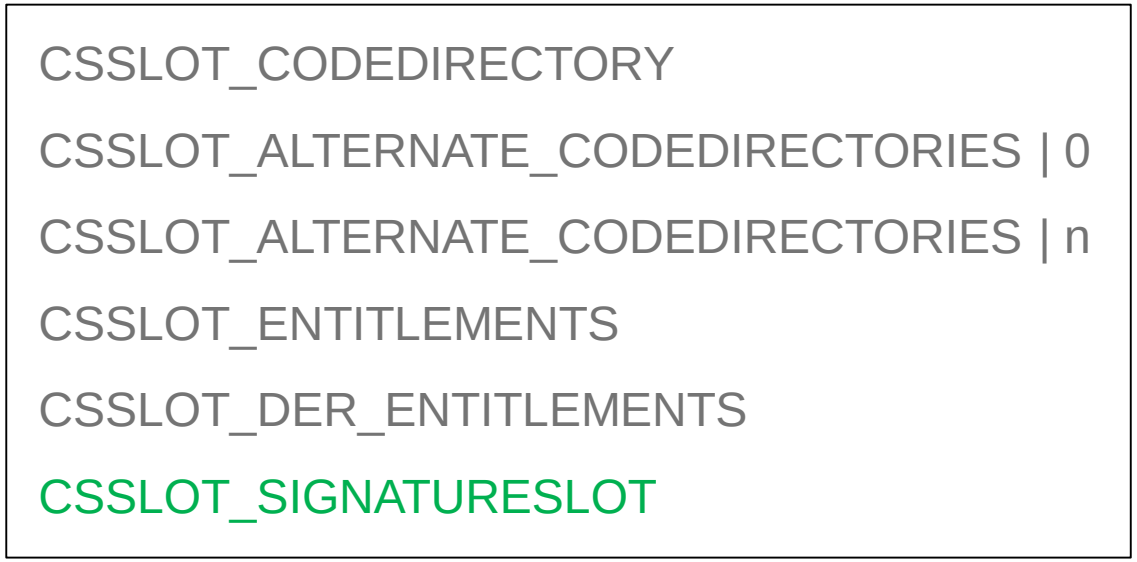
Code Signature Superblob



- Actual code directory that's valid for our binary
- Type: SHA256 (Kernel prefers SHA256 over SHA1)
- Nothing special about it, really

CVE-2023-41991: Signature Slot

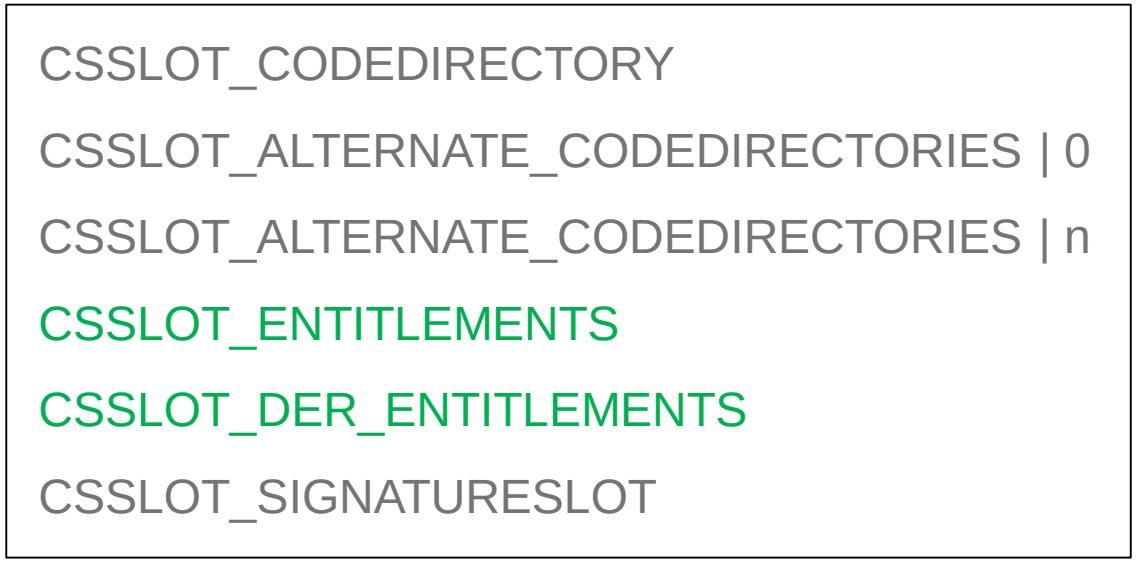
Code Signature Superblob



- Signer 1 (TrollStore certificate)
  - Our certificate, signed data controlled by us
  - Valid hash for main code directory (SHA1)
  - Valid hash for alternate code directory (SHA256)
- Signer 2 (Apple certificate)
  - Stolen from the same App Store app binary as the main code directory
  - Valid hash for main code directory (SHA1)
  - Invalid hash for alternate code directory (SHA256)

CVE-2023-41991: Entitlements

Code Signature Superblob



- Fully attacker controlled since their hashes are in the alternate code directory (SHA256) that we also fully control
- Gives you arbitrary permissions to do anything you want*
- *Except anything involving overtaking system processes, since these are isolated from the rest of the system and the system thinks we are an App Store app

POC


opa334
@opa334dev
Thanks, that worked! my artwork finally prints! 💯🔥
Post übersetzen
iPhone-13-Pro-von-Lars:~ root# print_artwork
zsh: killed print_artwork
iPhone-13-Pro-von-Lars:~ root# fastPathSign2 /var/jb/bin/print_artwork
iPhone-13-Pro-von-Lars:~ root# print_artwork
!!!! we out here! #AppStoreGang #FastPathSquad !!!!
iPhone-13-Pro-von-Lars:~ root#
12:02 vorm. · 26. Nov. 2023 · 237.955 Mal angezeigt

Big shoutout to @alfiecg_dev!!

TrollStore

- App-Installer that itself is signed with CoreTrust bug
- Gets root via persona-mgmt entitlement
- Accepts unsigned IPA files (apps) to be opened within it
- Applies CoreTrust bug on all executables in the app bundle
- Places app on the filesystem
- Adds it to the icon cache
- App appears on home screen and is usable like any other app

TrollStore vs. Jailbreak

- Persistent
- Only explicitly signed binaries can execute
- Not able to spawn launch daemons
- No system wide tweak injection

- Not persistent (unless chained with separate persistence bug)
- All unsigned binaries can execute
- Able to spawn launch daemons
- System wide tweak injection

Agenda

- Code Signing on iOS
- TrollStore 
- Dopamine 

Challenges of Modern Jailbreaks

- Kernel code is read only, enforced via hardware (KTRR)
- Some pointers are protected by pointer authentication (PAC)
- Some sensitive pages are protected by the Page Protection Layer (PPL)

Implementing a Modern Jailbreak

- Data-only
- Instead of hooking kernel code, hook userspace code
- Stash exploit primitives into a server, then offer various operations to clients (other processes)

- Assumptions
  - Kernel read/write primitive (acquired via exploit)
  - PPL bypass (required for codesigning bypass)

Implementing a Modern Jailbreak

- Kernel Exploit: kfd landa (CVE-2023-41974)
  - Supports iOS 15.0 - 16.6.1
- PPL Bypass: Operation Triangulation (CVE-2023-38606)
  - Supports iOS 15.0 - 16.5(.1)

- End result: Jailbreak for iOS 15.0 - 16.5 (all devices)

Code Signature Enforcement

- Every executables requires a valid code signature to run
  - System executables: Ad hoc signed, verified by CDHash in TrustCache
  - AppStore executables: Signed by Apple on submission using “Apple iPhone OS Application Signing” certificate
  - Xcode App executables: Signed by Developer certificate issued by Apple, extremely limited
- Checked by the Kernel on execution
  - During posix_spawn or execve syscall

Recall from earlier slide

Static TrustCache

CDHash(</sbin/launchd>)

CDHash(</usr/lib/dyld>)

CDHash(</usr/libexec/installd>)

(…)

- Linked list of arrays that contain trustworthy CDHashes
- Embedded in operating system
- Protected by KTRR 

Dynamic TrustCache

CDHash(</Developer/usr/bin/debugserver>)

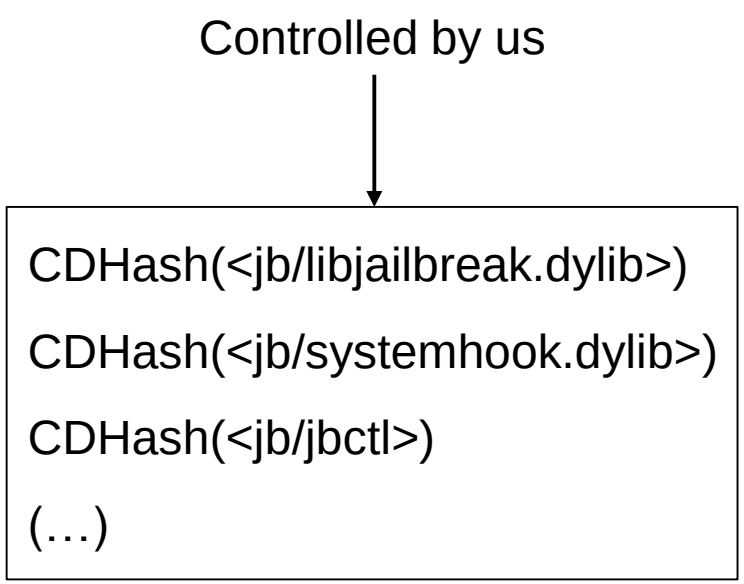
CDHash(</Developer/usr/lib/libsysmon.dylib>)

CDHash(</Developer/usr/libexec/sysmond>)

(…)

- Linked list of arrays of CDHashes from binaries inside Xcode debugging image
- Loaded at runtime when Xcode prepares debugger support
- Protected by PPL □

Bypassing Code Signing with PPL R/W



- Allocate our own TrustCache structure
- Insert it into the linked list
- Kernel now considers our executables trustworthy and allows them to be executed
- Libraries in TrustCache are allowed to be mapped system wide

Automatic Trust Caching

- Currently we can add files to TrustCache manually, but we want to fully bypass codesigning system wide (for all files)
- Idea
  - System tries to execute binary at path x or tries to map library at path x
  - Before being launched / mapped, CDHash of binary / library at path x is automatically added to TrustCache
  - How can we archive this in practice?

launchdhook.dylib

- Injected into launchd (pid 1) process at jailbreak time
- Manages PPL R/W primitives
- Provides ”jailbreak server” via mach and XPC, accessible system wide
- Also solves some other miscellaneous tasks, like loading third party launch daemons

systemhook.dylib

- Injected by launchdhook into every process spawned by launchd
- Reinjects itself into any other child process
- Hooks posix_spawn and execve syscalls to add the target CDHash to TrustCache by sending it to launchdhook via IPC

dyldhook

- Static patch of dynamic linker (dyld)
- Make DYLD_INSERT_LIBRARIES environment variable work
- Talks to launchdhook IPC to patch some stuff about the process and weaken the sandbox
- fcntl hook to make attaching any signature to a library work
  - Calculate CDHash of signature to be attached and send it to launchdhook, which will add it to TrustCache
  - Effectively disables library validation

posix_spawn hook

```
int
posix_spawn(pid_t *restrict pid, const char *restrict path,
    const posix_spawn_file_actions_t *file_actions,
    const posix_spawnattr_t *restrict attrp, char *const argv[restrict],
    char *const envp[restrict]);
```

- Send path to launchdhook, which will calculate the CDHash of the file and add it to TrustCache
- Modify envp (child process environment) to insert “DYLD_INSERT_LIBRARIES=systemhook.dylib”

Automatic Trust Caching: Summary

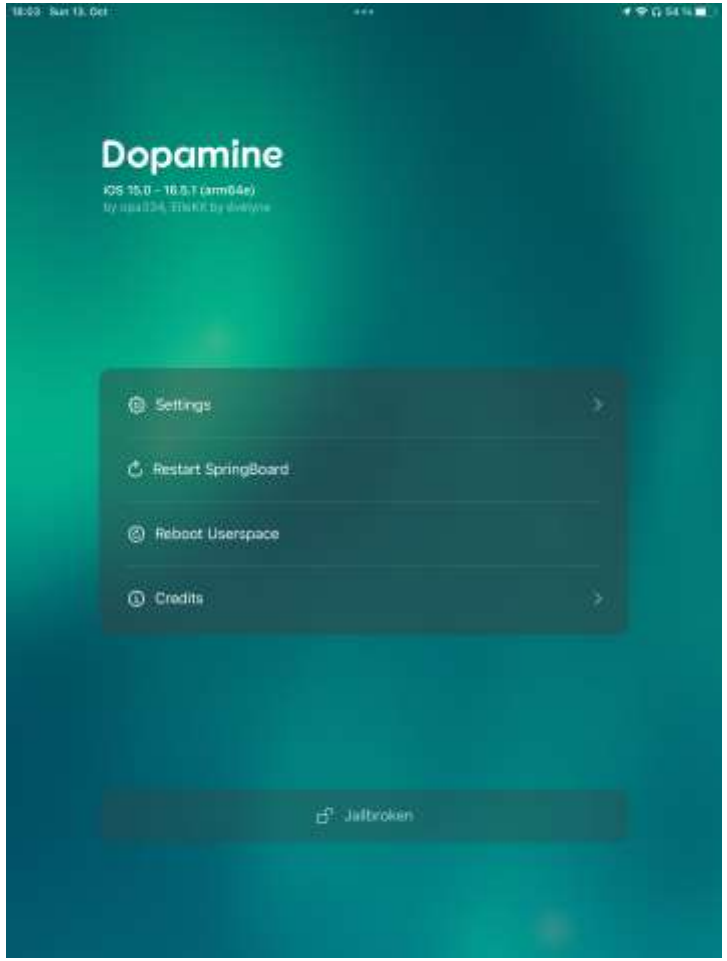
- In every process:
  - fcntl hook (in dyld) adds the CDHash of any library that’s about to be mapped to TrustCache
  - posix_spawn hook (in systemhook.dylib) adds the CDHash of any binary that’s about to be spawned to TrustCache

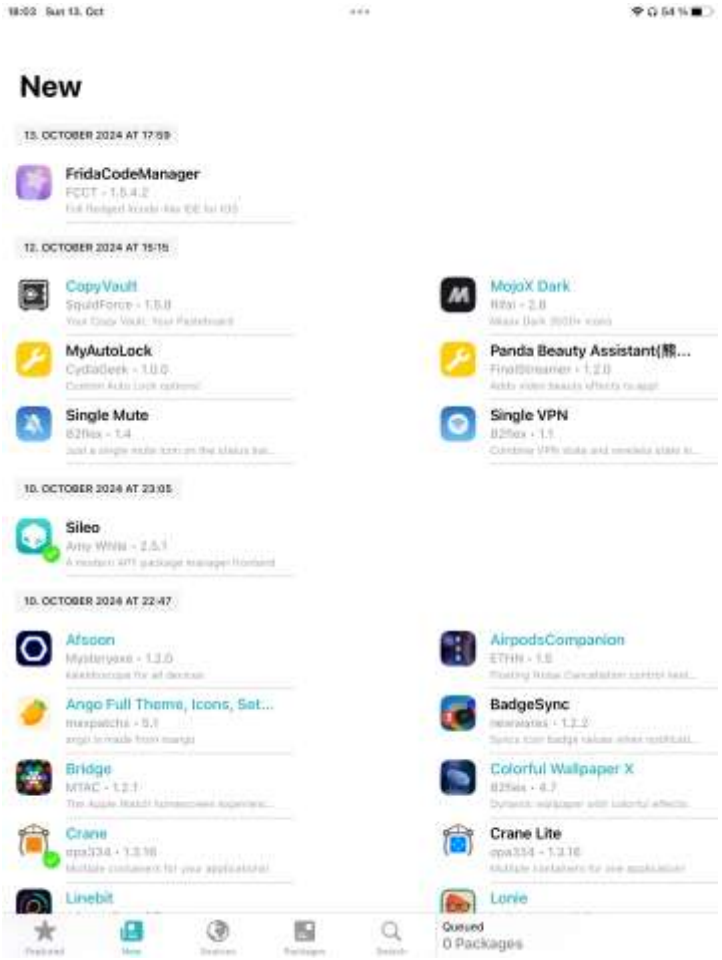
- Result: Code signing is bypassed; any binary can execute ✓

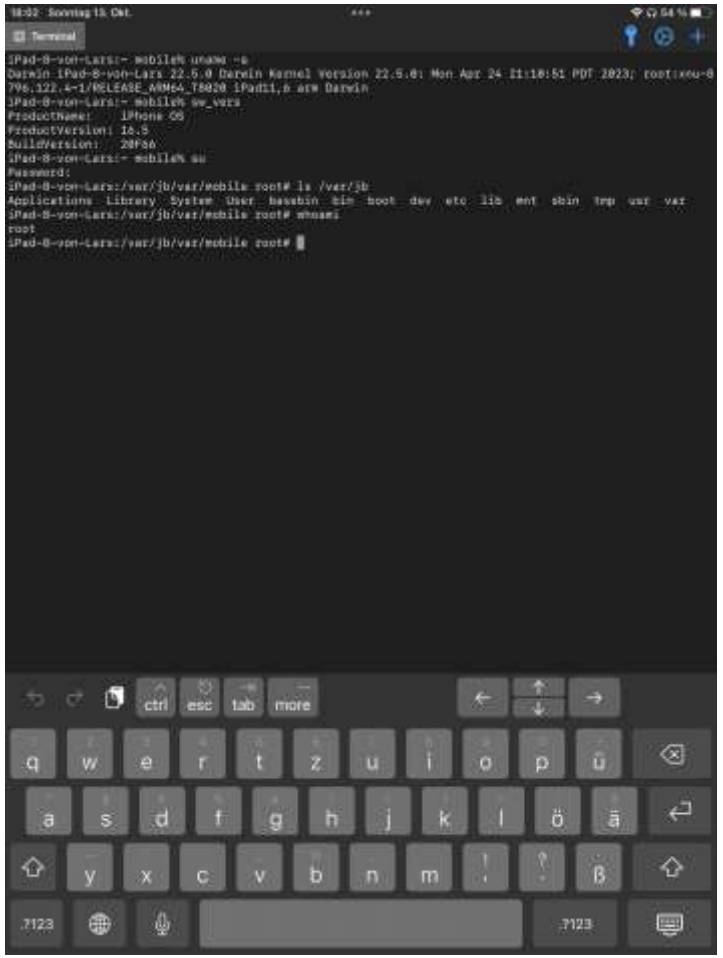
Enabling Tweak Injection

- In systemhook: dlopen(“/var/jb/usr/lib/TweakLoader.dylib”)
- External package will provide tweak loader library that takes care of any remaining logic
- Will parse third party extensions in “/var/jb/System/Library/MobileSubstrate/DynamicLibraries” and inject them as neccessary

Dopamine (iOS 15.0 - 16.5)







Wen eta iOS 17/18 jailbreak???

- Physical use after free bug class (used in kfd) killed in 17.3
- Root helpers (TrollStore’s “get root” method) killed in 18.0
- SPTM introduced in 17.0 (replaces PPL)

- No more public exploits
- Public kernel exploitation is as good as dead
- Eta never?

Thanks for your attention

Any questions?