Signed, Sealed and Delivered

Jonathan Levin

MacSysAdmin 2017 (1/2)

© 2017 Technologeeks.com,
NewOSXbook.com

Follow along

- http://NewOSXBook.com/files/jtool
  - Or http://NewOSXBook.com/tools/jtool.html

- http://NewOSXBook.com/files/MSA2017CS.pdf

Code Signing

- Applies digital signatures to executables
  - Ingredients:
    - Hash function (SHA-1 or SHA-256)
    - Private key (known to signer)
    - Public key (known to the world)
    - Certificate (authenticating public key as a “trusted” key)
  - Mach-O loader (in kernel) enhanced to validate signatures
    - Actually carried out in kernel and external extensions (via MACF)

Code Signing in Apple's OSes

- Apple introduced code signing as far back as OS X 10.5
  - In OS X, creeping in as of 10.8 (GateKeeper) and Mac App Store
  - In iOS, brought along with the App Store
- Other OSes can code sign too, but Apple is parsecs ahead:
  - Novel implementation, far more efficient than Linux or Android's
  - Provides a rich substrate for all of Darwin's system security measures
  - Enables Entitled binaries (and, indirectly, SIP)
  - Enables Code Requirements

Motivation for Code Signing

- Obvious motivation: Authenticate software origin
  - Greatly mitigates any potential for malware as Apple vets its Devs

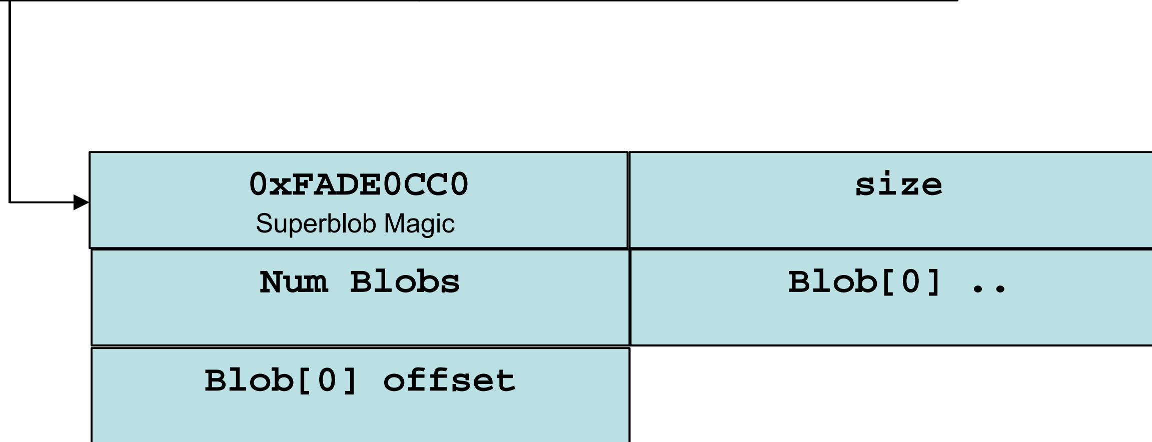
- Secondary motivation: Security profiles embedded in signature
  - OS X and iOS declarative security – entitlements – part of signature

- Unexpected bonus: Hegemony over software distribution
  - Only code signature allowed in iOS is Apple's.
  - OS X still allows any signature (or even unsigned code). For how long?

LC_CODE_SIGNATURE

- LC_CODE_SIGNATURE command points to a code signature "blob"
- Key component of blob is the "Code Directory"
  - Version: 0x20001 through 0x20400
  - Flags: Usually none, but "adhoc" (*OS) or others (see later)
  - Identifier: reverse DNS notation unique ID
  - CDHash: SHA-1/SHA-256 "mega-hash" of code slots
- All fields are big endian (in case PPC ever makes a comeback)
- Signature can also be "detached", (i.e. separate file)
  - Popular in iOS 11 for "removable" built-in Apps

LC_CODE_SIGNATURE	cmdsize
Dataoff (Offset of superblob from Mach-O header)	datasize (overall length of superblob)



0xFADE0C02
Code Directory Magic

0xFADE0C01
Requirements Vector Magic

0xFADE0C00
Single Requirement Magic

0xFADE0B01
CMS Signature Magic

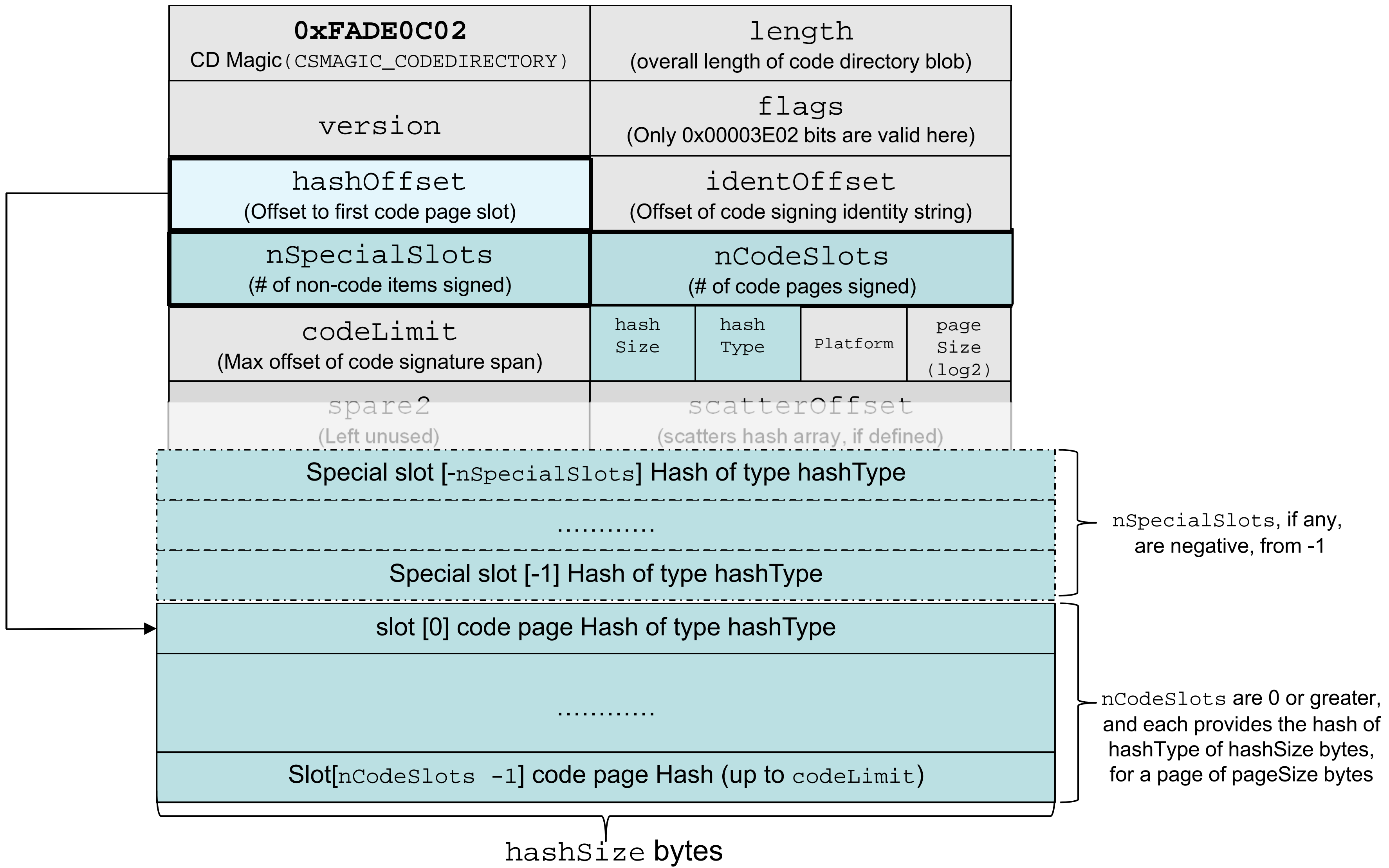
Version	XNU	Supports
0x20001	N/A	Modern features
0x20100	2422	ScatterOffset
0x20200	2782	TeamOffset
0x20300	3247	CodeLimit64
0x20400	iOS 11	Exec Segment

0xFADE0C02 CD Magic (CSMAGIC_CODEDIRECTORY)	length (overall length of code directory blob)
version	flags (Only 0x00003E02 bits are valid here)
hashOffset (Offset to first code page slot)	identOffset (Offset of code signing identity string)
nSpecialSlots (# of non-code items signed)	nCodeSlots (# of code pages signed)
codeLimit (Max offset of code signature span)	hash Size / hash Type / Platform / page Size (log2)
spare2 (Left unused)	scatterOffset (scatters hash array, if defined)
teamOffset (offset of Team Identifier)	spare3 (Left unused)
codeLimit64	
Exec Segment Base	
Exec Segment limit	
Exec Segment Flags	

Flags valid in Mach-O

Value	CS_ Flag
0x0002	ADHOC
0x0004	GET_TASK_ALLOW
0x0200	KILL
0x0400	CHECK_EXPIRATION
0x0800	RESTRICT
0x1000	ENFORCEMENT
0x2000	REQUIRE_LV

Hash Size/Type	Function
20/1	SHA-1
32/2	SHA-256


0xFADE0C02
CD Magic(CSMAGIC_CODEDIRECTORY)
length
(overall length of code directory blob)
version
flags
(Only 0x00003E02 bits are valid here)
hashOffset
(Offset to first code page slot)
identOffset
(Offset of code signing identity string)
nSpecialSlots
(# of non-code items signed)
nCodeSlots
(# of code pages signed)
codeLimit
(Max offset of code signature span)
hash Size
hash Type
Platform
page Size (log2)
spare2
(Left unused)
scatterOffset
(scatters hash array, if defined)
Special slot [-nSpecialSlots] Hash of type hashType
....
Special slot [-1] Hash of type hashType
nSpecialSlots, if any, are negative, from -1
slot [0] code page Hash of type hashType
....
Slot[nCodeSlots -1] code page Hash (up to codeLimit)
nCodeSlots are 0 or greater, and each provides the hash of hashType of hashSize bytes, for a page of pageSize bytes
hashSize bytes

Code Slots

- File pages are individually hashed into “slots”, at indices 0+
  - Choice of algorithm specified in CDHash “HashSize/Type”
  - MacOS < 12: SHA-1 MacOS >=12: SHA-256
- Ancillary data also hashed into “special slots”
  - Special slot have to occupy negative indices
  - Unused indices must be claimed if (abs) higher indices needed

Index	Contains
-1	Bound Info.plist (Manifest)
-2	Internal requirements
-3	Resource Directory (_CodeResources)
-4	Application Specific (largely unused)
-5	Entitlements (bound in code signature)

Example: Code signatures

- Apple's tool: codesign (-d –vvvvvvvvvvvvvv…..)

- jtool --sig

© 2017 Technologeeks.com, NewOSXbook.com

GateKeeper and Code Signatures

- Contrary to popular belief, GK doesn't enforce Code signing

Gatekeeper (Apple dramatization)

Gatekeeper (Real Life)

© 2017 Technologeeks.com, NewOSXbook.com

GateKeeper and Code Signatures

- Contrary to popular belief, GK doesn’t enforce Code signing
- If **started by launchd: (e.g. via Finder GUI)**
  - com.apple.quarantine xattr is respected
  - Quarantine.kext prevents unquarantining xattr
  - syspolicyd is consulted
  - Pop up is displayed to user
  - User chooses to approve (via System Preferences)
  - syspolicyd is updated
- If started from unquarantined (or uncaring) process:
  - Err.. OK

© 2017 Technologeeks.com, NewOSXbook.com

Enforcing code signatures

- Using sysctl:

```
root@Zephyr (~) #sysctl vm | grep cs_
vm.cs_force_kill: 0
vm.cs_force_hard: 0
vm.cs_debug: 0
vm.cs_all_vnodes: 0
vm.cs_enforcement: 0
vm.cs_enforcement_panic: 0
vm.cs_library_validation: 0
vm.cs_blob_count: 816
vm.cs_blob_size: 29667456
vm.cs_blob_count_peak: 1031
vm.cs_blob_size_peak: 35977312
vm.cs_blob_size_max: 8896512
```

- Try: sysctl vm.cs_enforcement=1

© 2017 Technologeeks.com, NewOSXbook.com

All your ~~bases~~ code ~~belong to~~ signed by us

- If code signing is enforced, signature MUST lead to Apple
  - Apple signs built-ins with Root/Apple Code Signing/Software Signing
  - Third party apps signed with Root/WWDR/MacOS App Signing
  - Provisioning profiles signed with Root/WWDR/…/[Dev/Ent] Profile

- Greatly reduces – but not elimitates - malware exposure
  - Lots of dev-signed malware coming through DMGs out there
  - APTs can infect existing processes via ROP or other

- Might disable innocent, but not Apple-compliant code
  - Read: procexp, jtool, and pretty much anything @NewOSXBook
    - Yours truly can't get a developer certificate ☹

© 2017 Technologeeks.com, NewOSXbook.com

csops[_audittoken]

```
int csops(pid_t pid, unsigned int  ops, void * buf, size_t size);
int csops_audittoken(pid_t pid, unsigned int  ops, void *buf,
                     size_t size audit_token_t * token);
```

Table 5-28: The various code signing operations (as of XNU 3247)

#	CS_OPS_ code	Purpose
0	_STATUS	Query code signing bits
4	_PIDPATH	Retrieve executable path (deprecated in 24xx)
5	_CDHASH	Retrieve Code Directory Hash
6	_PIDOFFSET	Retrieve text offset
7	_ENTITLEMENTS_BLOB	Retrieve entitlements blob
11	_IDENTITY	Retrieve code signing identity
10	_BLOB	Retrieve entire code signing blob
1	_MARKINVALID	Sets the invalid bit. This might lead to killing process
2	_MARKHARD	Sets the hard bit (does not kill)
3	_MARKKILL	Sets the kill-if-invalid bit
8	_MARKRESTRICT	Sets the restricted bit
9	_SET_STATUS	Sets multiple code signing bits simultaneously

© 2017 Technologeeks.com, NewOSXbook.com

csops[_audittoken]

- csops(2) allows various code signing operations:
  - Blob is retrieved from kernel space, therefore implicitly trusted.
- Thanks to csops(2), signatures provide a wide substrate:
  - Requirements define specific validation constraints
  - Entitlements: allow high level declarative permissions
- Code signature valiation built into procexp ('CS' column)
  - Simple test binary: http://NewOSXBook.com/tools/cs
  - Run with any PID as an argument to validate status and dump blobs

© 2017 Technologeeks.com, NewOSXbook.com

Entitlements

- Probably the most ingenious security mechanism ever

- Staggeringly simple:
  - XML plist with textual entitlements as declaratory permissions
  - Have entitlement = can perform operation
  - Don't have entitlement = bugger off (even as root!)

- Surprisingly effective:
  - XML plist is embedded in code signature **AND** signed
  - Provisioning profiles barred from arbitrarily entitling themselves
  - All other entitlements can only be signed directly by Apple.

© 2017 Technologeeks.com, NewOSXbook.com

Entitlements

- View any binary's entitlements using `jtool -ent`

```
morpheus@Zephyr (~) %jtool --ent `which 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>
```

- Retrieved programmatically by csops[_audit_token]
  - Validation is then a simple CFDictionaryGetValue()
  - SecTask*Entitlement* APIs provide simple interface
- Find a comprehensive, searchable database at

  http://NewOSXBook.com/ent.jl

© 2017 Technologeeks.com, NewOSXbook.com

Code signature requirements

- Can be specified as a (CSReq) blob in signature
  - Blob signed separately, in special slot -2

- Can also be validated on the fly:

```
OSStatus SecRequirementCreateWithStringAndErrors
  (CFStringRef text,
   SecCSFlags flags,CFErrorRef *errors,
   SecRequirementRef * __nonnull CF_RETURNS_RETAINED requirement);
```

- Verification fetches blob for kernel (csops)
  - Performs the rest in user mode.

- csreq(1) is an “expert tool for manipulating requirements”

© 2017 Technologeeks.com, NewOSXbook.com

Code signature requirements

AMFI Developer Requirement

(anchor apple generic and certificate leaf[field.1.2.840.113635.100.6.1.12] exists) or
(anchor apple generic and certificate leaf[field.1.2.840.113635.100.6.1.2] exists) or
(anchor apple generic and certificate leaf[field.1.2.840.113635.100.6.1.7] exists) or
(anchor apple generic and certificate leaf[field.1.2.840.113635.100.6.1.4] exists)

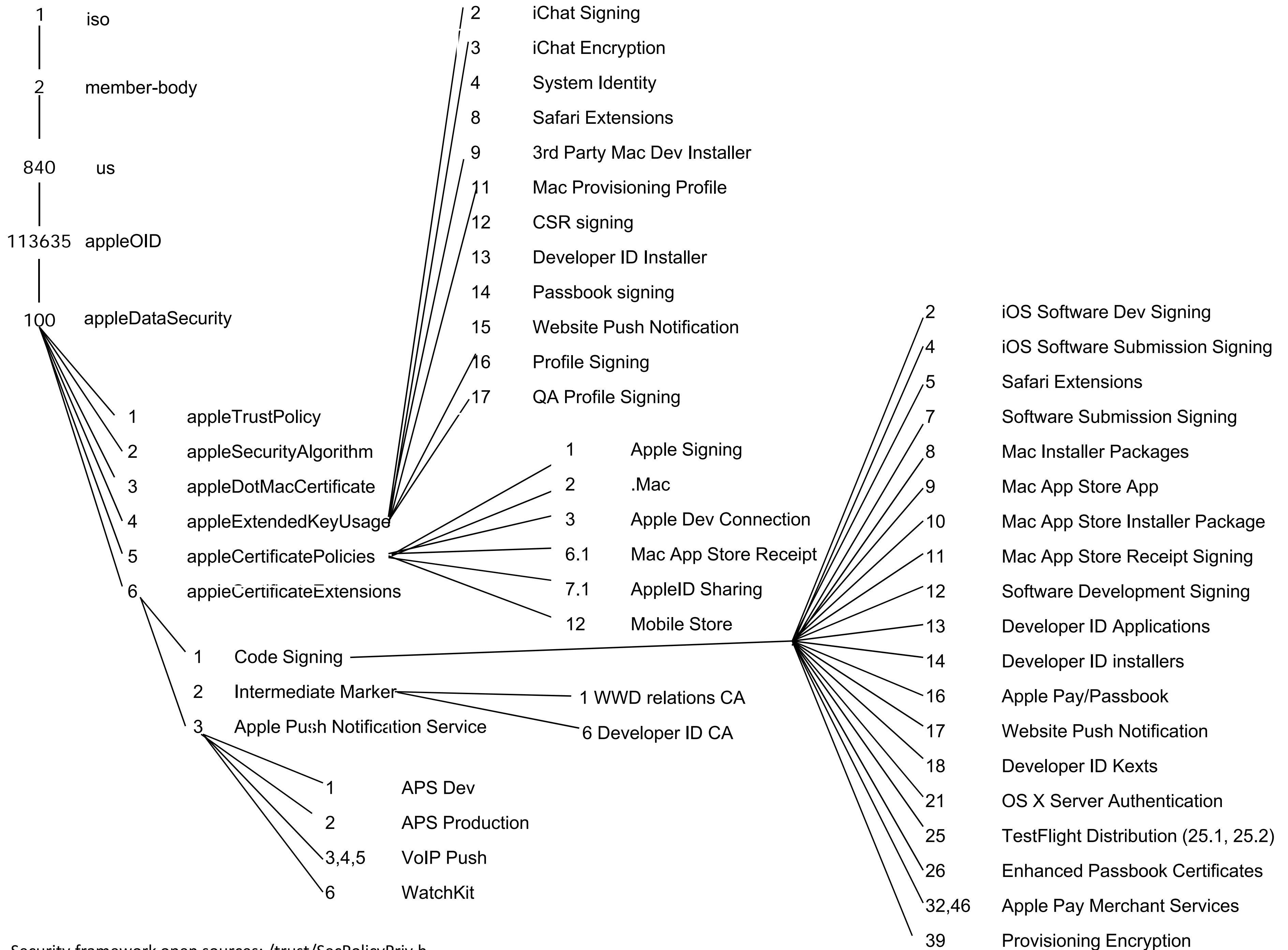
Specific exemption

anchor apple generic and certificate 1[field.1.2.840.113635.100.6.2.6] and
certificate leaf[field.1.2.840.113635.100.6.1.13] and
certificate leaf[field.1.2.840.113635.100.6.1.18] and
certificate leaf[subject.OU] = "6KR3T733EC"

AMFI Basic requirement:

(anchor apple) or
(anchor apple generic and certificate leaf[field.1.2.840.113635.100.6.1.9] exists) or
(anchor apple generic and certificate 1[field.1.2.840.113635.100.6.2.6] exists and
certificate leaf[field.1.2.840.113635.100.6.1.13] exists) or
(anchor apple generic and certificate leaf[field.1.2.840.113635.100.6.1.9.1] exists) or
((anchor apple generic and certificate leaf[field.1.2.840.113635.100.6.1.12] exists) or
(anchor apple generic and certificate leaf[field.1.2.840.113635.100.6.1.2] exists) or
(anchor apple generic and certificate leaf[field.1.2.840.113635.100.6.1.7] exists) or
(anchor apple generic and certificate leaf[field.1.2.840.113635.100.6.1.4] exists)

© 2017 Technologeeks.com, NewOSXbook.com



Security.framework open sources: /trust/SecPolicyPriv.h

Code signature requirements

AMFI Developer Requirement

(Software Development Signing) or
 (iOS Software Dev Signing) or
 (Software Submission Signing) or
(iOS Software Submission Signing)

Specific exemption

anchor apple generic and Developer ID CA
 and certificate Developer ID Applications and
Developer ID Kexts and
certificate leaf[subject.OU] = "6KR3T733EC"

AMFI Basic requirement:

(anchor apple) or
(Mac App Store App) or
(anchor apple generic and Developer ID CA and Developer ID Applications) or
(Mac App Store App.1) or (Software Development Signing) or
 (iOS Software Dev Signing) or
 (Software Submission Signing) or
(iOS Software Submission Signing)

© 2017 Technologeeks.com, NewOSXbook.com

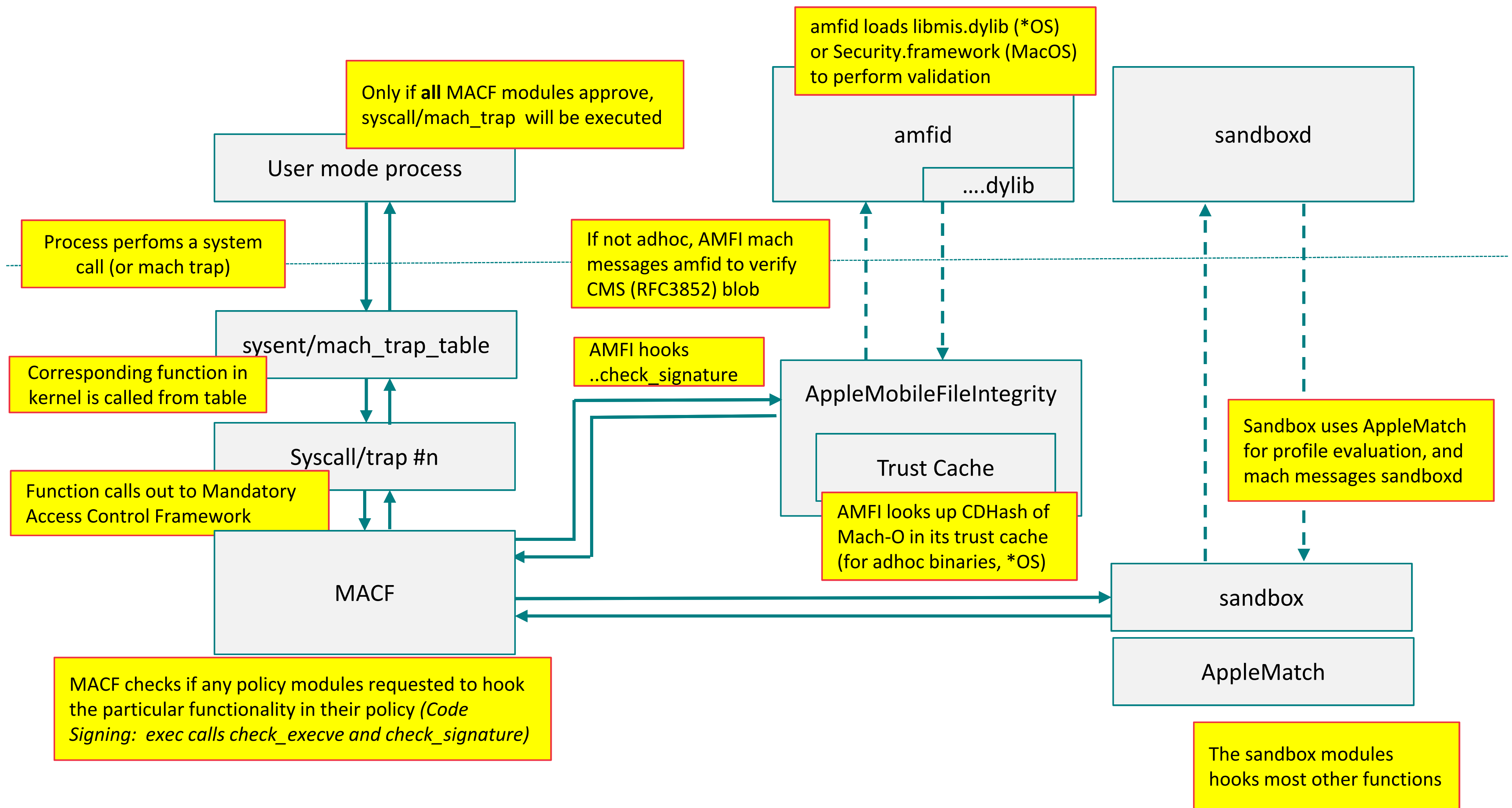
Code Signing: behind the scenes

- When a Mach-O binary is loaded:
  - Code signature blob and CodeDirectory is validated
  - Entire blob is stored in kernel's UBC

- On page fault:
  - Blob of binary is located in UBC
  - Corresponding page slot in CodeDirectory is retrieved
  - Faulting page is hashed accordingly
  - If hashes match, swell
  - If not:
    - CS_HARD: Page mapping fails, process may try to recover
    - CS_KILL: Process is killed on the spot with a SIGKILL.

© 2017 Technologeeks.com, NewOSXbook.com

The Unholy Trinity: MACF, AMFI & Sandbox



AppleMobileFileIntegrity.Kext

MAC policies have some 330+ callouts. AMFI cares about:

Callout	Called by MACF when:
mpo_cred_check_label_update_execve	MAC Label* needs to be updated as a result of process launching (exec)
mpo_cred_label_init/associate/destroy	MAC Label* lifecycle
mpo_proc_check_interit_ipc_ports	resets task/thread ports for setuid/setgid programs
mpo_proc_check_mprotect	mprotect(2) invoked (iOS prevents r-x from ever getting +w)
mpo_proc_check_map_anon	mmap(2) invoked with MAP_ANON
mpo_proc_check_get_task	task_for_pid trap (the holy grail of debugging/tracing/pwning) invoked
mpo_vnode_check_exec	exec(2) is invoked
mpo_proc_check_cpumon	CPU Usage Monitoring parameters
mpo_proc_check_run_cs_invalid	**Code Signature is invalid – AMFI gets a chance to save process**
mpo_vnode_check_signature	**Signature blob is added to Unified Buffer Cache**

* MAC Labels are used in the implementation of sandboxing – but that's for another presentation (and the book)

10.11 and rootless

- OS X 10.11/iOS 9 introduce “rootless” security
  – /System/Library/Sandbox/rootless.conf

- Root it still there, but restricted via default sandbox profile
  – /, /usr, /bin, /sbin now all protected from any modification

    (chflags on 10.11 will show “restricted”)

- Can be disabled (OS X), though only via recovery mode
  – Or via an in kernel call to csr_set_allow_all.

© 2017 Technologeeks.com, NewOSXbook.com

Rootless entitlements

Com.apple.rootless.*	Called by MACF when:
Install[.inheritable]	Bypass all filesystem checks
kext-management	Kextload like it's 2009
Restricted-block-devices	Access raw disks (/dev/disk*XXX* devices)
Restricted-nvram-variables	Access SIP configuration via csr-data NVRAM
Storage.*xxx*	Access files flagged with *xxx* in com.apple.rootless xattr
Volume.vm	Manage swap on a logical volume
Xpc.bootstrap	Push launchd(1) around (set up XPC domains and services)
Xpc.effective-root	(Nearly) Unlimited XPC power

© 2017 Technologeeks.com, NewOSXbook.com

Hark these words

- The day is near when Apple enforces cs by default
  – They did it for the *OS variants, and look what that got us
- First they came for root (SIP). Now for your developers
  – Devs will have no choice but to get a provisioning profile
- Overall, software security is likely to benefit from this
  – Software of unknown origins will be denied
  – Blacklisting will become far easier
- Not clear if this will put an end to malware
  – Unlikely – and will likely incite more sophisiticated malware.

© 2017 Technologeeks.com, NewOSXbook.com

Questions? Comments?

© 2017 Technologeeks.com, NewOSXbook.com