Hack in the (sand)Box

(The Apple Sandbox - five years later)

Jonathan Levin

http://NewOSXBook.com/
http://technologeeks.com
The Apple Sandbox

• Introduced way back in Mac OS 10.5 as “Seatbelt”
  – Very naive implementation originally, bypassed and opt-in

• Revamped in Mac OS 10.7 as “The App Sandbox”
  – Stronger implementation, introducing containers
  – Opt-in for Apple’s own binaries and apps
  – Mandatory for Mac App Store apps (but not for DMG based)

• Far stronger still in iOS
  – Mandatory for all third party applications
  – Evolved beyond MacOS implementation
## Sandbox versions

<table>
<thead>
<tr>
<th>Version</th>
<th>OS Version</th>
<th>Notable Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>..</td>
<td>OS X 10.5/iOS 1-3</td>
<td>Initial version, white list approach</td>
</tr>
<tr>
<td>..</td>
<td>OS X 10.6/iOS 4</td>
<td></td>
</tr>
<tr>
<td>165</td>
<td>OS X 10.7/iOS 5</td>
<td>Basic containers</td>
</tr>
<tr>
<td>220</td>
<td>OS X 10.8/iOS 6</td>
<td>Sandbox exceptions</td>
</tr>
<tr>
<td>278-300</td>
<td>OS X 10.9/iOS 7</td>
<td>IOKit get property, vnode renaming</td>
</tr>
<tr>
<td>358</td>
<td>OS X 10.10/iOS 8</td>
<td>Rootless (introduction, non-enforcing), get-task, AMFI integration (in OS X version), kexts (kind of)</td>
</tr>
<tr>
<td>460</td>
<td>OS X 10.11/iOS 9</td>
<td>Rootless enforcement, container manager</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Host special ports, kexts, OSX NVRAM finally protected</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Policy moved to __DATA.__const (iOS 9.2)</td>
</tr>
<tr>
<td>592</td>
<td>OS X 10.12/iOS 10</td>
<td>Container Manager enforcement (iOS)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>User data items</td>
</tr>
</tbody>
</table>
So Why Are We Here?

• Last actual research conducted in 2011:
  – Dionysus Balazakis seminal work - “The Apple Sandbox”

• Very little further research – partial, unpublished or both

• Sandbox has evolved by leaps and bounds
  – Further evolves in iOS 10 and MacOS 12
  – Provides “System Integrity Protection” as of MacOS 11 (not yet iOS)

• Provides first, strongest, and sometimes last line of defense
  – Tons of exploitable bugs in services and kexts blocked by sandbox
  – Breaking out of the sandbox is toughest stage of jailbreaking.
  – ... And eight of you here voted for this talk 😊
Plan

- Prerequisite: MACF
- MacOS (“App Sandboxing”)
- *OS (Containers)
- Reversing (MacOS, iOS implementations)
- Sandbox APIs

You’re welcome to follow along:
http://NewOSXBook.com/articles/hitsb.html
Prerequisite: MACF

• Mandatory Access Control Framework serves as substrate
  – XNU’s implementation modeled after FreeBSD’s
  – Compare - SELinux/SEAndroid

• Simple idea, powerful impact:
  – Kernel extensions provide a “policy” and call mac_policy_register
  – Policy contains “hooks” (callbacks)
  – Depending on process label, callbacks get invoked
  – Kernel extension gets to inspect operation arguments
  – Return 0 to allow, non-zero to thwart operation
  – All registered hooks must allow operation.
MACF Policy Hooks

User mode process

Process performs a system call (or mach trap)

sysent/mach_trap_table

Corresponding function in kernel is called from table

Syscall/trap #n

Function calls out to Mandatory Access Control Framework

MACF

MACF checks if any policy modules requested to hook the particular functionality in their policy

Policy Module

Policy module inspects operation, and allows or denies

Policy module may perform upcalls to user mode lackey

User Mode Daemon

Only if all MACF modules approve, syscall/mach_trap will be executed

Additional Policy Module...
MACF Policy Modules

• Serves as basis for virtually all of Apple’s OS Security
• Currently 5 known policy modules:

<table>
<thead>
<tr>
<th>Kext</th>
<th>Oses</th>
<th># Ops</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quarantine</td>
<td>MacOS</td>
<td>~15-17</td>
<td>Gatekeeper. Sort of.</td>
</tr>
<tr>
<td>MCXALR</td>
<td>MacOS</td>
<td>1</td>
<td>Managed Client Extensions (MDM/Parental Controls)</td>
</tr>
<tr>
<td>TMSafetyNet</td>
<td>MacOS</td>
<td>~26</td>
<td>TimeMachine hooks on file access</td>
</tr>
<tr>
<td>AMFI</td>
<td>All (OSX &gt;=10.10)</td>
<td>~8-13</td>
<td>Enforce code signing, some entitlements &amp; Mach ports</td>
</tr>
<tr>
<td>Sandbox</td>
<td>All</td>
<td>130+</td>
<td>Confine, strangle and block Applications at every turn</td>
</tr>
</tbody>
</table>

• Labels can define which policy, if any, will take effect
  – Process can be execed into label with `mac_execve(#380)`
  – `posix_spawnattrs` can similarly enforce sandbox
  – Sandbox has own spawnattrs (for specific container or profile)
Sandboxing

• Original sandbox approach – “seatbelt” – opt in:
  – You’d have to ask to confined (like, want to go to jail!)
  – Like its namesake, most people find it borderline troublesome.

• Contemporary sandbox approach is radically different:
  – You are either containerized or you are not:
    • Voluntary: because you are a responsible developer
    • Semi-voluntary: Code signature or location (Apple controlled)
    • Non-voluntary: Based on install location (*OS)
  – If containerized, Sandbox intercepts all important operations
    • Definition of important keeps increasing to include more..
    • Operation assessed versus a profile, or entitlements
MacOS : App Sandboxing

- Sandbox no longer requires sandbox_init – but signature
  - This way Apple, not developer, can enforce sandboxing
  - In iOS, /var/mobile/Containers/Bundle location auto-sandboxes

- In MacOS, com.apple.security.app-sandbox sandboxes

- com.apple.application-identifier for container
  - Otherwise defaults to CFBundleIdentifier from App’s Info.plist

- com.apple.application-groups (~10.7.5, 10.8.3 and later)
  - ~/Library/Group Containers/…
1) Process loads libSystem.B

2) libSystem Initializer calls libsecinit

3) Libsecinit registers with securityd

4) Securityd decides whether or not process needs to be sandboxed

5) If decision is affirmative, libsecinit voluntarily imposes sandbox on process
MacOS : App Sandboxing

- Containers created at ~/Library/Containers/{CFBundleIdentifier}
- All Structured the same way:
  - Container.plist: metadata (in bplist00 format)
    - Identity (Unicode, Base64)
    - Compiled profile (SandboxProfileData, base64)
    - SandboxProfileDataValidationInfo (long dict...)
    - Version (36 = MacOS 10, 38 = MacOS 11, 39 = MacOS 12)
  - Data: Directory structure, mimicking user’s home directory:
    - .CFUserTextEncoding
    - Documents
    - Library
    - Music
    - Desktop
    - Downloads
    - Movies
    - Pictures
MacOS : App Sandboxing

• Data directories are often symbolic links(!)
  – SandboxProfileDataValidationRedirectablePathsKey limits links

• Metadata also holds entitlements, and other parameters
  – SandboxProfileDataValidationEntitlementsKey
  – SandboxProfileDataValidationParametersKey
iOS: Containers

Also allows for shared containers
  - Apps with same team-id can share data
iOS 10 Containers

iOS 10 continues the evolution of containers, by once again moving Application static data to /var/containers, leaving /var/mobile/Containers with just Data/ and Shared/. The Application/ sub-directory structure has also been chown(2)ed to _installd. This is likely in anticipation of full multi-user capabilities.

Listing `xx-conta`: Containers in iOS 8-9 vs. those in 10

```
/var/mobile
+-----+ Containers
    +-----+ Bundle
    |     +-----+ Application
    |     +-----+ Framework
    |     +-----+ PluginKitPlugin
    |     +-----+ VPNPlugin
    +-----+ Data
    |     +-----+ Application
    |     +-----+ InternalDaemon
    |     +-----+ PluginKitPlugin
    |     +-----+ TempDir
    |     +-----+ VPNPlugin
    |     +-----+ XPCService
    +-----+ Shared
        +-----+ AppGroup
/var/containers
    +-----+ Bundle _installld
    +-----+ Application
    +-----+ Framework
    +-----+ PluginKitPlugin
    +-----+ TempDir
    +-----+ VPNPlugin
    +-----+ Data root:wheel
    +-----+ Shared root:wheel
    +-----+ SystemGroup ACLs
```

Another interesting change in iOS10 is the inclusion of a new `SystemGroup/` shared container, which uses for the first time Access Control Lists (ACLs), as shown in Output `xx-1`:

Output `xx-contacts`: Access Control Lists on the Shared/SystemGroup containers

```
# As of iOS 10, shared system group containers also have ACLs
 drwx-xr-x+ 3 root wheel 136 Jul 7 12:40 /var/containers/root /le_shared/SystemGroup/
 0: allow list,add_file,search,delete,add_subdirectory,delete_child,readattr,writeattr, readattrr,readattr,readsecurity,writesecurity,chown, file,inherit, directory,inherit, only_inherit
 1: allow add_file,add_subdirectory,readattrr,writeattr

 drwx-xr-x+ 3 root wheel 136 Jul 7 12:40 systemgroup.com.apple.plist,.msinfo
 0: allow list,add_file,search,delete,add_subdirectory,delete_child,readattr,writeattr, readattrr,readattr,readsecurity,writesecurity,chown, file,inherit, directory,inherit, only_inherit
 1: allow add_file,add_subdirectory,readattrr,writeattr
```
iOS: Containers

- The sandboxd has been entirely removed in iOS as of 9.x
  - Still used in MacOS, primarily for tracing

- New daemon – `containermanagerd` – takes over
  - Part of `MobileContainer` private framework
  - Communicates with user mode (installld, etc) over XPC port
  - Communicates with kernel mode (kext) over Special Port #25
    - MIG message 0x13392fd4 (322514900)
    - Contains sb_packbuff payload of kernel requests
AMFI

- Sandbox and AMFI make good bedfellows
- AMFI ensures signature, provides entitlement services
- Sandbox depends on AMFI (as of 358 in MacOS)

iOS Sandbox uses specific entitlements:

- seatbelt-profiles – assign a particular profile to binary
- com.apple.private.security.container-required - Sandboxes built-in apps
Deconstructing Sandbox

• MacOS Sandbox.kext can serve as a good reference
  – Largely same codebase, with some differences, but symbolicated

• Joker can auto-symbolicate plenty*: 
  – Stubs to kernel functions
  – Entire MACF Policy (120+ functions!)

• Can get other functions (no names, yet) with jtool: 
  – grep BL.*0x | cut -dx -f2 then feedback to companion file 
  – About 150 additional functions revealed by this method

• Important functions (e.g. smalloc, sfree) yield rest.
  – Hook_policy_syscall especially important (for mac_policy_syscall)

* - Joker 3 can now handle split kexts from XNU 3750+!
Sandbox MACF Policy Hooks

- Most MACF Policy hooks call cred_sb_evaluate
  - 1st argument (in R0/X0/RDI) is MACF's
  - 2nd argument (in R1/X1/ESI) encodes operation number
Sandbox MACF Policy Hooks

• Operation numbers correspond to hard-coded names
  – Can also be found in older libsandbox.1.dylib
    • Removed (precompiled) into 570+
  – Names can be found in kext’s __DATA__CONST.__const
    • Not going away since they are needed for APIs
  – There are more operations than there are MACF hooks
    • Some are callable from user mode by apps (e.g. AppleEvents, TCC)
Sandbox MACF Policy Hooks

- **cred_sb_evaluate** calls **sb_evaluate**
  - 1st parameter is sandbox obtained from label_get_sandbox
  - Operation as 2nd Parameter
  - Buffer as 3rd Parameter

```assembly
// Disassembling from file offset 0x7f5d, Address 0xfffffffff006b96c70 to next function

PUSH {X2,X1}
MOV X19, X20
LDR X0, [X20, #16]
ADD X29, SP, #32
MOV X20, X2
MOV X21, X1
MOV X20, X2
MOV X21, X1
MOV X20, X2
MOV X21, X1
MOV X20, X2
MOV X21, X1
MOV X20, X2
MOV X21, X1
MOV X20, X2
MOV X21, X1
RET
```
Sandbox MACF Policy Hooks

- cred_sb_evaluate derives credentials, and calls eval
  - May or may not report sandbox violations (based on argument to check)

  ```
morpheus@Zephyr (../10) % filetool -d sb_evaluate com.apple.security.sandbox.kext | grep BL
Opened companion File: ./com.apple.security.sandbox.kext.ARM64.D6145CC4-1EDA-34AF-A613-A0E613FE791F
Disassembling from file offset 0x79ec0, Address 0xfffffffff006b9ee0 to next function
ffffffff006b9eeec BL _derive_cred ; 0xfffffffff006b9f0fc
ffffffff006b9ef20 BL _kauth_cred_proc_ref.stub ; 0xfffffffff006ba81cc
ffffffff006b9ef3c BL _OSCompareAndSwapPtr.stub ; 0xfffffffff006ba7f44
ffffffff006b9ef48 BL _kauth_cred_unref.stub ; 0xfffffffff006ba81d8
ffffffff006b9ef70 BL _eval ; 0xfffffffff006b9f164
ffffffff006b9efbc BL _eval ; 0xfffffffff006b9f164
ffffffff006b9f010 BL _derive_vnode_path ; 0xfffffffff006b9e590
ffffffff006b9f030 BL _derive_socket_info ; 0xfffffffff006ba1030
ffffffff006b9f070 BL _sb_trace ; 0xfffffffff006ba1f0c
ffffffff006b9f0bc BL _sb_report ; 0xfffffffff006ba161c
ffffffff006b9f0cc BL _free_filter_context ; 0xfffffffff006b9e410
morpheus@Zephyr (../10) %
```

- Evaluation first attempted against platform_profile
- Can default to specific process-defined (container) profile

* - MacOS implementation slightly different (includes csr_check, etc). iOS also inlines eval_filter into eval
Reversing Profiles

- Sandbox Profiles are written in tinyScheme (UGH!)
  - In MacOS – plaintext, in /System/Library/Sandbox/Profiles
    - Per framework profiles also exist for Apple’s frameworks
  - in iOS – compiled & built-in!

- The gist:
  - (version 1) (only version supported)
  - (deny default) (least privilege)
  - (allow …….) (selectively allow APIs)
  - (deny …….) (selectively disallow APIs)

- Can apply and trace using sandbox-exec:

  (version 1)
  (trace "/tmp/appTrace.sb")
Sandbox-exec

• Simple binary (300-500 lines of ASM)

```
morpheus@Zephyr (~)$ sandbox-exec
Usage: sandbox-exec [options] command [args]
Options:
  -f profile-file     Read profile from file.
  -n profile-name     Use pre-defined profile.
  -p profile-string   Specify profile on the command line.
  -D key=value        Define a profile parameter.
Exactly one of -f, -n, -p must be specified.
```

• MacOS 11 adds undocumented “-t” for tracing
  – Tracing broken in iOS with the removal of sandboxd 😞

• Closed source – but....
  – Will also dump compiled profile in /tmp
  – Provides first implementation of sandbox-exec for iOS!
Built-in Profiles

• MacOS originally had 4 “built-in” profiles
  – Weren’t so useful in the first place and largely deprecated

• iOS extends that to dozens of profiles
  – Can be found in kext
  – Can also be found in iOS’s libsandbox.1.dylib
    • AGXCompilerService ... wifiFirmwareLoader

• Built-in profiles are precompiled
  – Originally, maintained by sandboxd
  – In iOS 9+, maintained inside kext (__TEXT.__const)

```
morpheus@Zephyr (%/10) %jtool -v -d __TEXT.__const com.apple.security.sandbox.kext | grep -A 4 -a A.G.X.C
Opened companion File: ./com.apple.security.sandbox.kext.ARM64.D6145CC4-1EDA-34AF-A613-A0E613FE791F
Dumping from address 0xffffffff00630e620 (Segment: __TEXT.__const) to end of section
Address : 0xffffffff00630e620 = Offset 0x520
0xffffffff006353360: 13 00 00 00 41 47 58 43  A G X C
0xffffffff006353368: 6f 6d 70 69 6c 65 72 53  compilerS
0xffffffff006353370: 65 72 76 69 63 65 00 00  service
0xffffffff006353378: 0e 00 00 00 48 6f 6e 65  H/de
0xffffffff006353380: 76 2f 70 74 6d 78 0f 00  v / ptmx
```

(C) 2012, 2016 Technologeeks.com
Containerizing Applications (iOS)

- **mpo_cred_label_update_execve hook**: MACF calls sandbox, because it registered hook
- **amfi_copy_seatbelt_profile_names**: Get `<seatbelt-profiles>` entitlement, if any
- **PE_I_can_haz_debugger**: Allows debug_mode (if boot arg was set, non-issue)
- **get_signing_identifier**: Gets application-identifier (from entitlement)
- **get_container_required_entitlement**: Get `com.apple.private.security.container-required` ent
- **check sandbox spawnattrs**: Several validations on spawn attributes
- **Upcall to container manager**
Validator container name

No null bytes, special cases for plugins, keyboard, etc..

```
builtin_sandbox_create
```

Create a sandbox object, from a builtin (precompiled) profile

```
platform_set_container
```

Create com.apple.sandbox.container Sandbox Extension

```
Create executable extension
```

Allow App to access and launch own executable

```
cred_set_sandbox
```

Attach sandbox struct to kauth credentials

```
Revoke privileged ports
```

Remove access to dangerous Mach Ports
Sandbox APIs

• Sandbox usermode APIs provided by two libraries:
  – `/usr/lib/system/libsandbox.sandbox.dylib`
    • Re-exported by LibSystem.B.dylib
    • Mostly direct APIs to kext
  – `/usr/lib/libsandbox.1.dylib`
    • Profile compilation
    • TinyScheme implementation statically linked in
    • Plenty of Scheme strings/profile definitions in __TEXT.__const

• Containment (often) performed over `mac_execve()`
• KEXT APIs invoked over `macf_syscall()`
Sandbox APIs

• mac_syscall (#381) used extensively:
  – Allows ioctl(2) style multiplexing of syscalls provided by a kext
  – Generic mechanism, used by all policy modules
  – On kext end, hook_policy_syscall enables multiplexing
  – Different offerings in MacOS and *OS
Sandbox APIs

- Syscall implementations differ in between OSes, versions!

<table>
<thead>
<tr>
<th>Op</th>
<th>Sandbox function</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>_set_profile[_builtin]</td>
<td>Set a profile (=label &amp; containment) of a process</td>
</tr>
<tr>
<td>2</td>
<td>_check</td>
<td>Check if operation is allowed in confines of sandbox</td>
</tr>
<tr>
<td>3</td>
<td>_note</td>
<td>Attaches a note (memory buffer) to sandbox (offset 0x80)</td>
</tr>
<tr>
<td>4</td>
<td>_container_path_for_pid</td>
<td>Retrieve container path for a given PID</td>
</tr>
<tr>
<td>5-7</td>
<td>_extension_issue/consume/release</td>
<td>Issue, apply and remove a temporary exception</td>
</tr>
<tr>
<td>8-9</td>
<td>_extension_update_file[_with_new_type]</td>
<td>Update/twiddle extension</td>
</tr>
<tr>
<td>10-11</td>
<td>_suspend/unsuspend</td>
<td>Suspend/resume sandbox checks for PID*</td>
</tr>
<tr>
<td>13-15</td>
<td>_policy_syscall related...</td>
<td>iOS, routed to container manager</td>
</tr>
<tr>
<td>16</td>
<td>_inspect</td>
<td>Dump tons of great information on SB.</td>
</tr>
<tr>
<td>17</td>
<td>profile_dump</td>
<td>Dumps compiled profile for a PID (MacOS, AppleInternal** 😵)</td>
</tr>
<tr>
<td>19</td>
<td>_vtrace[enable</td>
<td>disable</td>
</tr>
<tr>
<td>21</td>
<td>_rootless_allows_task_for_pid</td>
<td>Does current policy allow task_for_pid call?</td>
</tr>
</tbody>
</table>

- Get a more accurate list with jtool’s switch detection (ARM64)

* - Don’t get excited. Process can only do it on itself, if entitled as a sandbox-manager *and* another exception entitlement..
** - csr_check(0x01) – can be tweaked via direct access to NVRAM
Sandbox APIs

• **sandbox_check** especially useful:
  – Widely used in tweaks to gauge sandbox restrictions
  – Commonly used with SANDBOX_CHECK_NO_REPORT
    • Performs check silently, without any user-mode output

• Really useful for probing container XPC/file restrictions
  – Much more reliable than decompiling!
• Sandbox 570+ adds **sandbox_check_bulk**
Demo: sbtool

```
root@Padishah (/var/root)# ps -ef | grep MobileSafari
  501  3427  1  0 Mon06AM ??    0:00.44 /Applications/MobileSafari.app/MobileSafari
root@Padishah (/var/root)# sbtool 3427 mach | grep No | head -5
Checking Mach services for 3427....
com.apple.Preferences.gsEvents: Nope
com.apple.mobilemail.gsEvents: Nope
com.apple.timezoneupdates.tzd.server: Nope
com.apple.streaming_zip_conduit: Nope
com.apple.pfd: Nope
root@Padishah (/var/root)# sbtool 3427 mach | grep Yep | head -5
Checking Mach services for 3427....
com.apple.voiceservices.tts: Yep
com.apple.nehelper: Yep
com.apple.coremedia.videocompositor: Yep
com.apple.coremedia.mutablecomposition: Yep
com.apple.managedconfiguration.mdmpush-prod: Yep
```
Sandbox APIs - undocumented

- sandbox_inspect_pid super useful, but undocumented:
  - Available in *OS as of somewhere in 460 (iOS 9.something)
  - Implemented via __sandbox_ms (... 0x10);
  - Very valuable information on process, directly from kext

- Requires root privileges (or AppleInternal build)
Demo: sbtool

root@Padishah (/var/root)# sbtool 3427 inspect
MobileSafari[3427] sandboxed.
size = 439166
container = /private/var/mobile/Containers/Data/Application/607D2C61-76F7-49AF-B3FB-B6B4BE45AA47
sb_refcount = 210
profile = container
profile_refcount = 56
extensions (3: class: com.apple.sandbox.executable) {
    file: /Applications/MobileSafari.app (unresolved); flags=0
}
extensions (5: class: com.apple.sandbox.system-container) {
    file: /private/var/containers/Data/System/738391BB-914B-4AEF-88CE-D8758754CCBD (unresolved); flags=0
}
extensions (5: class: com.apple.security.exception.mach-lookup.global-name) {
    mach: com.apple.mobile.keybagd.xpc; flags=0
    mach: com.apple.parsec.subscriptionservice.internal; flags=0
    mach: com.apple.SafariCloudHistoryPushAgent; flags=0
    mach: com.apple.Safari.SafeBrowsing.Service; flags=0
}
extensions (7: class: com.apple.security.exception.files.absolute-path.read-only) {
    file: /private/var/mobile/Library/Caches/com.apple.storeservices (unresolved); flags=0
}
extensions (8: class: com.apple.sandbox.container) {
    file: /private/var/mobile/Containers/Data/Application/607D2C61-76F7-49AF-B3FB-B6B4BE45AA47 (unresolved); flags=0
}
Sandbox Extensions

Extensions allow exceptions to a given profile

iOS apps get the “standard extensions”:
- com.apple.sandbox.executable
- com.apple.sandbox.container
- com.apple.sandbox.application-group
Sandbox Extensions

Apple’s App provide even more extensions for themselves:

<table>
<thead>
<tr>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.apple.security.exception.shared-preference.read-write</td>
</tr>
<tr>
<td>com.apple.sandbox.application-group</td>
</tr>
<tr>
<td>com.apple.tcc.kTCCServiceAddressBook</td>
</tr>
<tr>
<td>com.apple.sandbox.executable</td>
</tr>
<tr>
<td>com.apple.app-sandbox.read</td>
</tr>
<tr>
<td>com.apple.security.exception.mach-lookup.global-name</td>
</tr>
<tr>
<td>com.apple.security.exception.iokit-user-client-class</td>
</tr>
<tr>
<td>com.apple.security.exception.files</td>
</tr>
<tr>
<td>com.apple.sandbox.container</td>
</tr>
</tbody>
</table>
Sandbox Extensions

- Before sandboxing, caller can set extensions (unless forbidden)

- Extensions are issued by sandbox kext as “tokens”
  - Hmac_sha1 with secret value (not exposed to user space)
Take Aways

• If you’re even loosely interested in OSX/iOS:
  – The sandbox is the first, possibly last line of security
  – In iOS, provides the most important obstacle to jailbreaking
  – In MacOS, containerizes AppStore Apps, and implements SIP

  – Source of sandbox_exec clone
  – Sbtool – open source
  – Ongoing documentation on profile reversing
  – Fully symbolicated companion file for iOS 10 kext
Suggested Links

  – Volume III (Security & Insecurity) available for pre-order!

• http://NewOSXBook.com/forum - Open forum for MOXiI

• http://Technologeeks.com/OSXRE - Related Training