Intro to macOS Security

by Carlos Polop

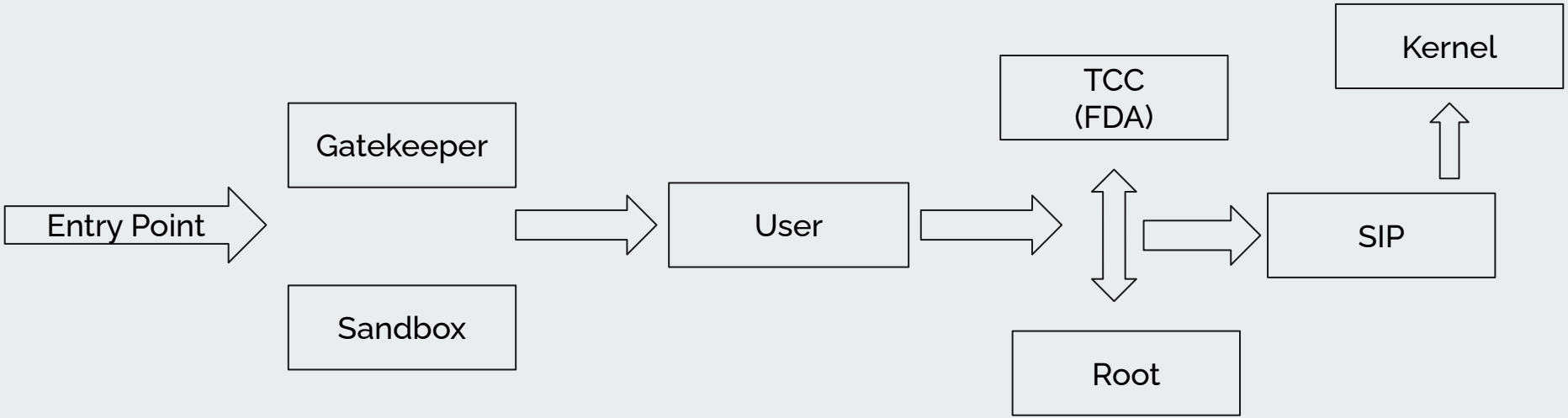
ls /Users/

@hacktricks_live

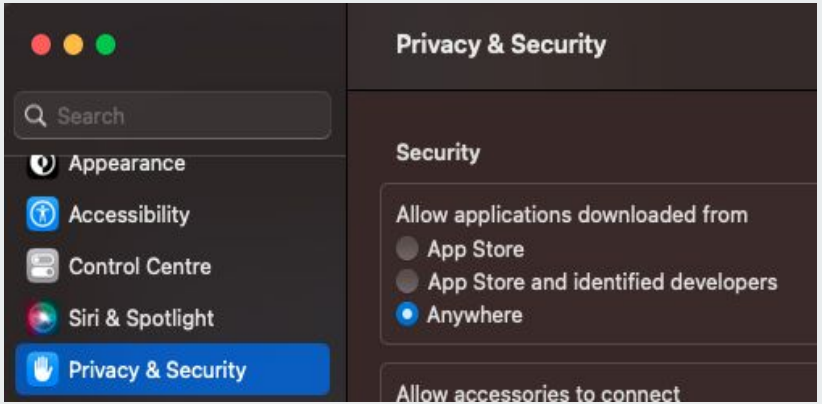
CARLOS POLOP

- Cloud, Infra & WebApp Principal Security Engineer/Team Lead at Halborn.
- Pentester/Red Teamer. Several certifications.
- Captain of the Spanish team in ECSC2021 & member of Team Europe in ICSC2022.
- Author of Hacktricks & PEASS-ng.

macOS Compromission Path



Gatekeeper



- Notarization
  - Apple checks for malicious content
  - Code signing
  - Needed for publishing apps in the App Store
- Gatekeeper
  - Config which apps are allowed
  - CLI tool: spctl
  - Rules db: /var/db/SystemPolicy
- Quarantine xattr
  - Indicates the file is suspicious (downloaded from the Internet, created by a sandbox process…)
  - Triggers Gatekeeper
  - Added by the software saving the file
    - Although with a flag in Info.plist is automatically added
- XProtect
  - Check for malware by:
    - Hashes
    - BundleID
    - TeamID
    - Yara rules

Gatekeeper - Bypasses

CVE-2021-1810

When extracted by **Archive Utility**, file **paths longer than 886** characters would fail to inherit the com.apple.quarantine extended attribute, making it possible to **bypass Gatekeeper for those files**.

CVE-2021-30990

Automator applications execute what is inside `application.app/Contents/document.wflow` not in the main executable. Therefore, it's possible to **point with a symbolic link to another Automator Application Stub inside the system (**`/System/Library/CoreServices/Automator\ Application\ Stub.app/Contents/MacOS/Automator\ Application\ Stub`**)** and it will execute what is inside `document.wflow` (you script) **without triggering Gatekeeper** because the actual executable doesn't have the quarantine xattr.

Prevent adding the quarantine attribute

At this time, if the ".app" folder doesn't have the quarantine attribute, Gatekeeper isn't going to be triggered

CVE-2022-42821

The ACL **`writeextattr`** can be used to prevent anyone from writing an attribute in a file.
Moreover, **AppleDouble** file format copies a file including its ACEs.
In the **source code** it's possible to see that the ACL text representation stored inside the xattr called **`com.apple.acl.text`** is going to be set as ACL in the decompressed file. So, if you compressed an application into a zip file with **AppleDouble** file format with an ACL that prevents other xattrs to be written to it... the quarantine xattr wasn't set into the application:

```
chmod +a "everyone deny write,writeattr,writeextattr" /tmp/test
ditto -c -k test test.zip
python3 -m http.server
```

Already inside

Just remove the quarantine flag to prevent Gatekeeper from checking the app

```
xattr -d com.apple.quarantine output.app
```

Sandbox

- Apps with the entitlement com.apple.security.app-sandbox are applied /System/Library/Sandbox/Profiles/application.sb profile
  - Any app from the App store
- Apps with the entitlement com.apple.security.temporary-exception.sbpl can have a custom sandbox definition
- Working directories of sandboxes can be found in ~/Library/Containers
- Everything created/modified by a sandboxed process will have the quarantine attribute.

Sandbox limits:

- Files/folders (read/write/create)
- Network access
- Binary execution
- IPC communication
- ...

Sandbox Bypass

Bypass Gatekeeper

If you can bypass Gatekeeper, you could execute an unsandboxed app.

Abusing Open

It's possible to use open to ask Launchd to launch an application (outside of the sandbox). It was possible to use the open params --env and --stdin to poison the env vars or stdin and execute arbitrary code outside of the sandbox.

- Those arguments are now ignored if launched from sandbox, but open can still be used

Abusing Auto-Startup locations

- If the sandbox allows to write inside a path where a file will be executed (cron, plist, plugins...)
- It's possible that the new process has another sandbox

Bypass Quarantine xAttr

If you are able to create an arbitrary folder without the quarantine attribute, it's possible to escape the sandbox.

- Any file created or modified from a sandboxed app will get the quarantine xattr
- Example: Mount a devfs volumen as it doesn't support xattrs

Preventing Sandbox Load

The process sandbox itself by loading libSystem.B (calling libSystem.B.initializer). If the binary prevents this library for being loaded, or hook any of the responsible functions to sandbox itself, the sandbox won't be applied:

- Function hooking with DYLD_INSERT_LIBRARIES
- Static binaries don't load that library (check in App Store)

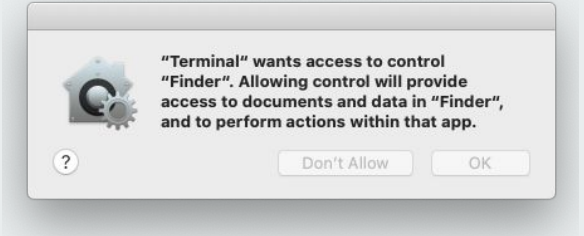
TCC & Entitlements

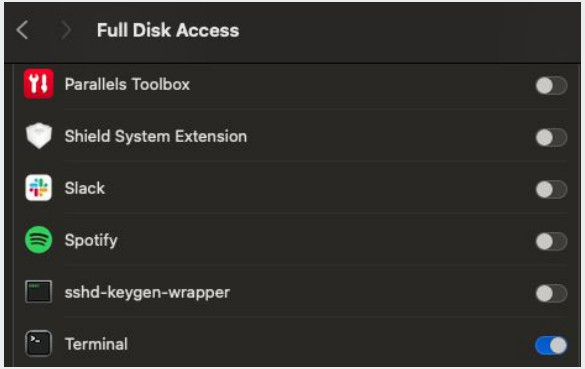
- **Limits** access to Documents, Photos, Messages, sensitive files in /etc…
  - Even root is limited
  - Access given to apps and not to users

- **Databases** (the process need FDA to read it):
  - /Library/Application Support/com.apple.TCC/TCC.db
  - $HOME/Library/Application Support/com.apple.TCC/TCC.db

- **Full Disk Access** (FDA): Basically, access to everything the user has access to
  - FDA and other permissions granted inside the System TCC database are shared across users
  - Still limited by SIP

- **Entitlements:** Permissions granted to apps in the form of strings
  - `codesign -vvv -d --entitlements - /System/Applications/TV.app`






```
[Key] com.apple.private.tcc.allow
[Value]
        [Array]
                [String] kTCCServiceAddressBook
                [String] kTCCServicePhotos
                [String] kTCCServiceAppleEvents
                [String] kTCCServiceSystemPolicyAllFiles
                [String] kTCCServiceCamera
```

TCC Bypasses

- **Write Bypass**: Even if you cannot read Desktop, you can write on it.
- **TCC Bypass** by absolute path
- **SSH Bypass:** Will usually have FDA granted
- Compromise $HOME env var: Create your own user database (`$HOME/Library/Application Support/com.apple.TCC/TCC.db`) and escalate to FDA
- **Mount**: It was possible to mount a disk over the TCC database to control it.
- **Injection:**
  - **Code/Command injection:** Abuse the TCC permissions of the victim process
  - **Child process Injection:** Inherit the TCC permissions of the victim process
    - Plugins
    - Scripts
    - Parameters
    - En vars

TCC Privilege Escalation

- **From user TCC DB to FDA:**
  - Abusing Automation privilege to abuse Finder (which has FDA privs)
- **From FDA to user TCC DB bypass:**
  - Overwrite users TCC DB
- **From SIP bypass to TCC bypass:**
  - Remove SIP protection of /Library/Application Support/com.apple.TCC/TCC.db
  - Add an exception in /Library/Apple/Library/Bundles/TCC_Compatibility.bundle/Contents/Resources/AllowApplicationsList.plist
- **From FDA to root*:**
  - CVE-2020-9771

Proc Injection - Library Validation

- If the library and binary are signed with the same certificate
- If the binary has one of the entitlements:
  - `com.apple.security.cs.disable-library-validation`
  - `com.apple.private.security.clear-library-validation` (depends)
- If the binary doesn't have the hardened runtime flag (every app in the store) or the library validation flag.

Proc Injection - DYLD_INSERT_LIBRARIES

- Abusing env variable DYLD_INSERT_LIBRARIES (like LD_PRELOAD in linux)
- Only works if:
  - Not setuid/setgid
  - No __RESRICT section in the macOS binary
  - No hardened runtime flag
  - No Library Validation flag
  - No restrict flag (statically or dynamically)
  - Sufficient: The app has the entitlement:

    `com.apple.security.cs.allow-dyld-environment-variables + No Library validation`

Proc Injection - Electron Applications (I)

- Electron Fuses:
  - **RunAsNode:** If disabled, it prevents the use of the env var ELECTRON_RUN_AS_NODE to inject code.
  - **EnableNodeCliInspectArguments:** If disabled, params like --inspect, --inspect-brk won't be respected. Avoiding his way to inject code.
  - **EnableEmbeddedAsarIntegrityValidation:** If enabled, the loaded asar file will be validated by macOS. Preventing this way code injection by modifying the contents of this file.
  - **OnlyLoadAppFromAsar:** It will only check and use app.asar (and not other file names), thus ensuring that when combined with the embeddedAsarIntegrityValidation fuse it is impossible to load non-validated code.
  - **LoadBrowserProcessSpecificV8Snapshot:** If enabled, the browser process uses the file called browser_v8_context_snapshot.bin for its V8 snapshot.

Proc Injection - Electron Applications (II) - Demo

- ELECTRON_RUN_AS_NODE
  - `ELECTRON_RUN_AS_NODE=1 /Applications/Discord.app/Contents/MacOS/Discord`
  - `require('child_process').execSync('/System/Applications/Calculator.app/Contents/MacOS/Calculator')`
  - If the fuse RunAsNode is disabled the env var ELECTRON_RUN_AS_NODE will be ignored, and this won't work.
- NODE_OPTIONS
  - `require('child_process').execSync('/System/Applications/Calculator.app/Contents/MacOS/Calculator');`
  - `NODE_OPTIONS="--require /tmp/payload.js" ELECTRON_RUN_AS_NODE=1 /Applications/Discord.app/Contents/MacOS/Discord`
  - If the fuse EnableNodeOptionsEnvironmentVariable is disabled, the app will ignore the env var NODE_OPTIONS
  - New error: Most NODE_OPTIONs are not supported in packaged apps. See documentation for more details.
- Inspecting
  - `/Applications/Signal.app/Contents/MacOS/Signal --inspect=9229`
  - `require('child_process').execSync('/System/Applications/Calculator.app/Contents/MacOS/Calculator')`
  - If the fuseEnableNodeCliInspectArguments is disabled, the app will ignore node parameters (such as --inspect)

Proc Injection - Chromium - Demo

- From https://twitter.com/RonMasas/status/1758106347222995007: Chromium apps can use:
  - --load-extension → To load extensions
  - --use-fake-ui-for-media-stream → Grant media stream permissions (without prompts)
- It's possible to:
  - Inject JS code in pages
  - Dump cookies
  - Keylogger in the browser
  - Capture browser traffic
  - …
- Works in:
  - Edge
  - Chrome
  - Opera
  - Brave
  - Chromium

Child Injection - Java Applications

_JAVA_OPTIONS

- Similar to NODE_OPTIONS, it allows to add new java parameters to use when a Java app is executed.
- Crash the app to execute a script or load an agent (careful with Java versions!)
- Find Java apps with: sudo find / -name 'Info.plist' -exec grep -l "java\." {} \; 2>/dev/null

vmoptions

- File with more Java parameters that Java processes load

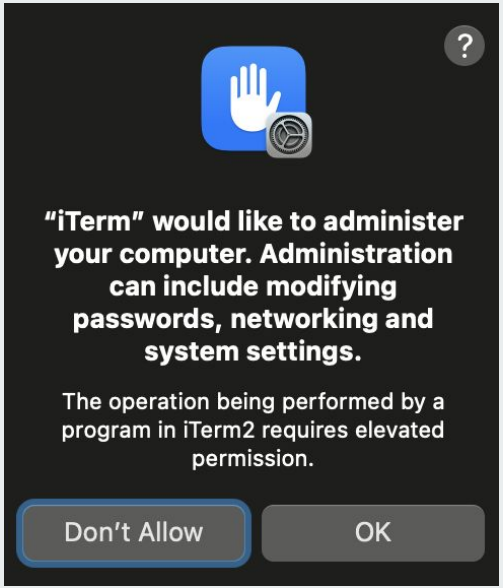
Child Injection - Perl - Demo

PERL5OPT & PERL5LIB

- Apple still sues some Perl scripts
- Use PERL5OPT to indicate which module force to load when perl is executed (or even indicate commands to run)
- Use PERL5LIB to indicate a new path to search for Perl module

Dependencies Hijacking

- Search dependencies with: perl -e 'print join("\n", @INC)'
  - /Library/Perl/5.30 (writable by root), not SIP protected like others
- Hijack a common used library in this folder.
- TCC message will appear:
  - Still useful for high privilege bypasses like SIP



Child Injection - bash/zsh

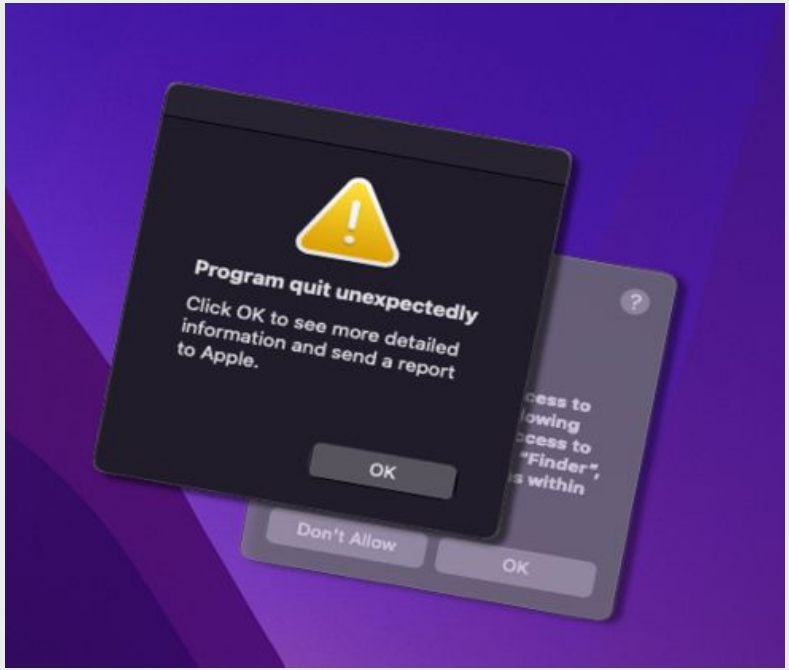
- Most applications, sooner or later, will run cli commands.
- Programming languages by default call sh/bash/zsh to run a command.
- Env variable BASH_ENV or file paths like ~/.bashrc, ~/.zshrc, ~/.zshenv, /etc/zshenv… are loaded and executed.
    - So it's possible to inject arbitrary commands in the process

Child Injection - Python/Homebrew

- Couldn't find any way to make python execute arbitrary code via env variables
- Couldn't hijack the path python built-in libraries even if their library code is writable
  - Python auto load built-in modules (it doesn't read the path os.__file__ to load it)
- Most python installations use Homebrew:
  - Adds writable and unprotected /opt/homebrew/bin to PATH
  - Path injected at the beginning of PATH
  - Possible to hijack python3 and other binaries
- Can try to get broader reach adding a controlled folder inside /etc/paths so root PATH is also compromised
  - Requires root but not TCC protected
- Hijack common binaries:
  - cat
  - mv
  - cp
  - sleep
  - sudo
  - …
- When running "/usr/bin/sudo" PATH is maintained → Binaries Hijacking == Privilege Escalation

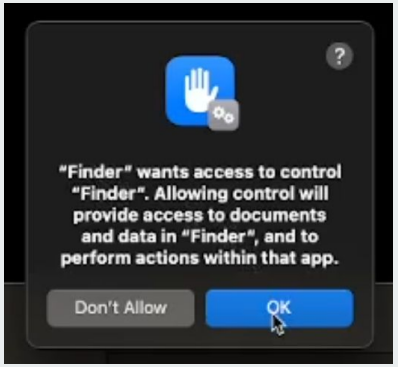
TCC ClickJacking

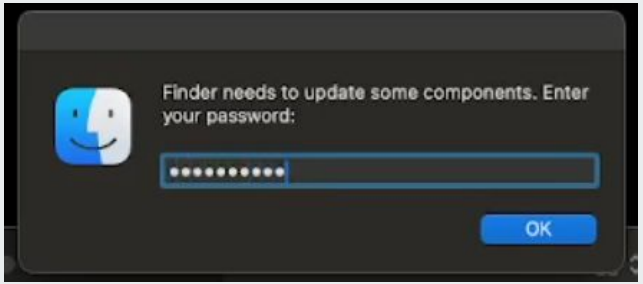
- It's possible to add a window over the TCC window to make the user allow a TCC prompt without visualizing it
  - PoC: https://github.com/breakpointHQ/TCC-ClickJacking



App Impersonation TCC Privesc

- Create an app with the name of a known one
- Use the real icon
- Set it in the Dock so the user launches it
- Make the app:
  - Ask to control Finder (privesc)
  - Get the password of the user
- It works with third party (Google Chrome) and apple (Finder) apps





Privilege Escalation

- **From user to root**
  - In macos, default user belongs to admin group
- CVEs, privesc misconfigurations (LinPEAS / MacPEAS)
- **FDA to root*: CVE-2020-9771**
  - **Any user**
  - An app with **Full Disk Access** (max privilege in TCC)
    - FDA permissions across users
  - Create a Time Machine Snapshot
  - Mount it
  - Read access over ALL THE DISK (even root and other users files)
  - Still working…

Easy sudo Privilege Escalation

PATH hijack

- When sudo is execute, $PATH is maintained
- Hijack common commands

Sudo Hijack

- Poison a file like ~/.zshenv with a new PATH
- Hijack sudo
- Make sudo execute your command and the original command

SIP - System Integrity Protection

- Also called rootless
- Prevents:
  - Modifying certain files (indicated in /System/Library/Sandbox/rootless.conf)
  - Debugging Apple processes (also kernel)
  - Modifying NVRAM variables
  - Loading Kernel extension
  - …
- If bypassed:
  - Write malware that cannot be deleted
  - Load kernel extensions
  - Bypass TCC by overwriting the System DB
- Entitlements:
  - com.apple.rootless.install.heritable
  - com.apple.rootless.install

SIP - Bypasses

- Shrootless

system_installd (com.apple.rootless.install.heritable) run post-install scripts during the installation of an Apple-signed packages. It loaded commands from /etc/zshenv

- CVE-2022-22583

system_installd was using /tmp to store the post-install script in a location protected by SIP... but /tmp isn't protected by SIP, so it was possible to mount a new folder.

- Mount over SIP folders

It was previously discovered that SIP was bypasseable by mounting folder over SIP protected directories

- fsck_sc

Create a symlink from /dev/diskX to /System/Library/Extensions/AppleKextExcludeList.kext/Contents/Info.plist and use fsck_sc (whose entitlements allow it to write over the protected file) to write garbage logs over the plist, to make it useless and make it possible to load the kernel extension AppleHWAccess.kext.

- systemmigrationd

systemmigrationd can bypass SIP and executes bash and perl scripts and can be abused via BASH_ENV and PERL5OPT env vars... Exposed in DEF CON 31 talk (2023)

- macOS Installers

Can bypass these protections

Launch/Environment Constraints

4 types of constraints:

- Self Constraints: Where the process is being launched from
  a. e.g. on-system-volume → The binary is inside the (unwritable) system volume
- Parent Process Constraints: What the parent process is
  a. e.g. is-init-proc → The parents executable is launchd
- Responsible Constraints: How the process is being called
  a. e.g. Restriction of the process calling a XPC communication
- Load Library Constraints:
  a. e.g. Indicate the libraries TeamID allowed to be loaded

Apple binaries has different categories (combinations of these constraints) restricting each Apple binary.

Environment Constraints are the option to add constraints to third party applications.