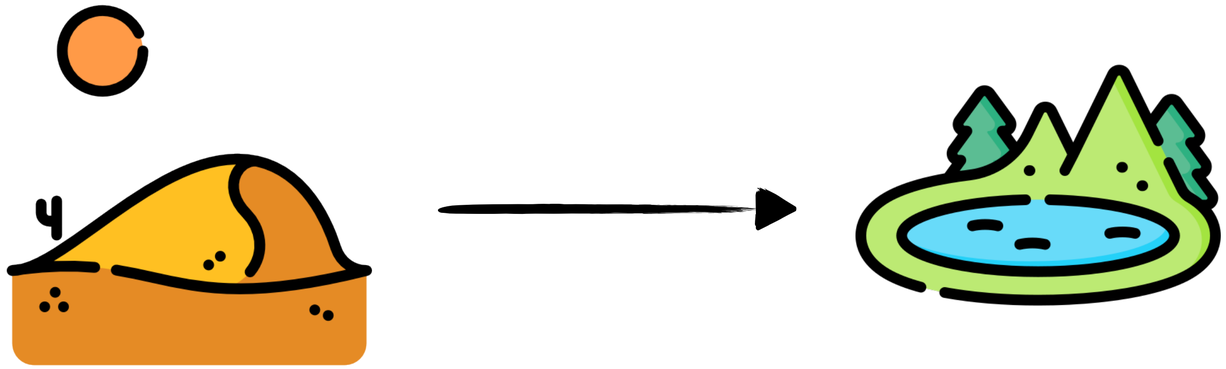
Evolution of macOS security from the Desert to the Lake



kandji

Csaba Fitzl
Twitter: @theevilbit

whoami

- Principal macOS Security Researcher @Kandji
- author of EXP-312 - macOS Exploitation training (🐙) at OffSec
- ex red/blue teamer
- macOS bug hunter (~100 CVEs)
- husband, father
- hiking, trail running 🥾 🏔️

Not what you expect...

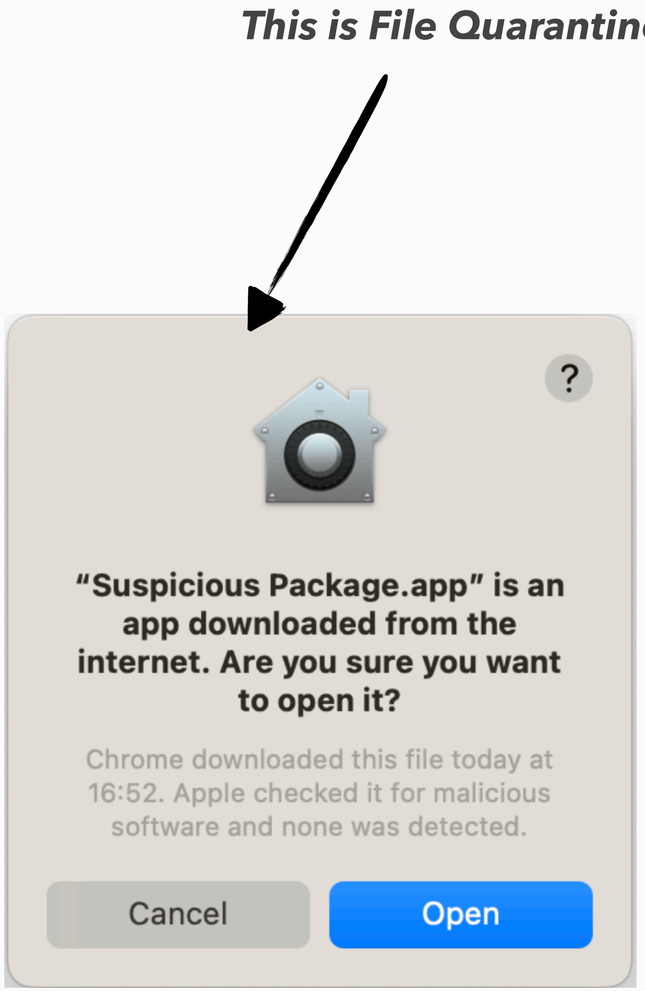
agenda

1. GateKeeper improvements
2. KEXT mitigations
3. TCC improvements
4. Process Injection Mitigations
5. Launch Constraints
6. Closing Weaponization Paths

GateKeeper

Some Terms

- GateKeeper <> GateKeeper
- 3 different technologies:
  - File Quarantine
  - GateKeeper
  - XProtect



What are they?

- File Quarantine (Mark Of The Web on the evil "W")
  - Downloaded apps need user consent to run
  - Always invoked on first execution
- GateKeeper
  - Verifies code signature, and ensures it conforms to set policy
  - Can be disabled
- XProtect
  - Checks against known malware

This is File Quarantine

Pre-Mojave

- Mac OS X 10.5 Leopard (2007): File Quarantine
- Mac OS X 10.6 Snow Leopard (2009): XProtect
- Mac OS X 10.7 Lion (2011): spctl command line
- Mac OS X 10.8 Mountain Lion (2012): Launch of Gatekeeper
  - Mac App Store
  - Mac App Store and identified developers
  - Anywhere

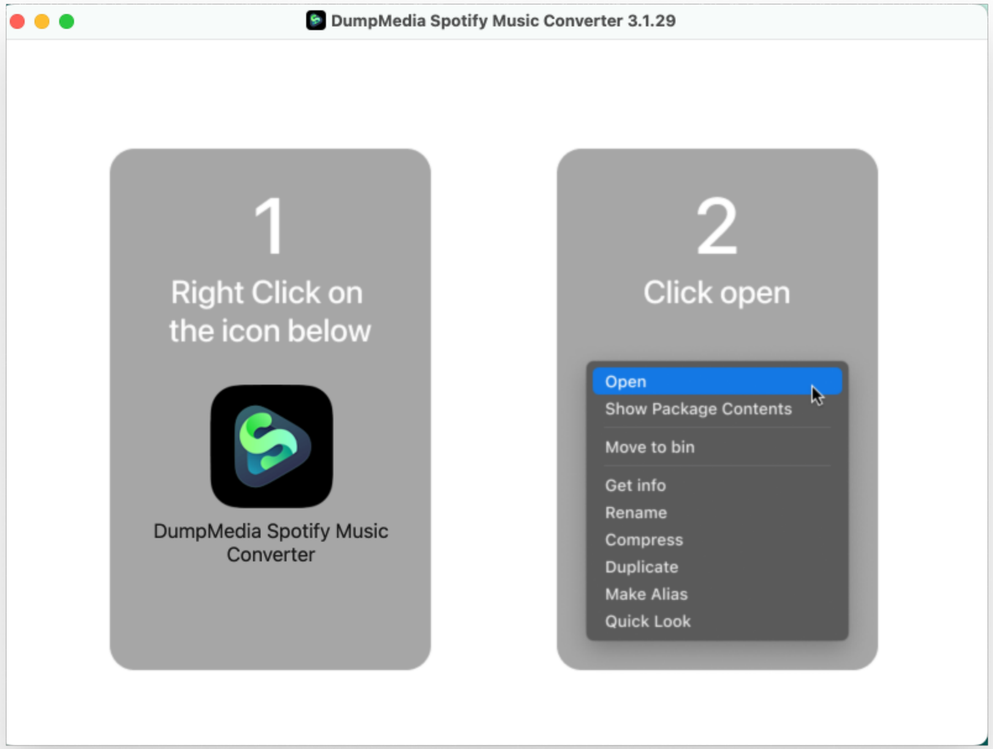
macOS 10.14 - Mojave

- Only integrated into LaunchServices
- Trivially to "bypass" via exec

```
chmod +x m
./m
```

macOS 10.15 - Catalina

- MAJOR change: integrated into spawn / exec
- Introduction of notarization
- users can bypass with right-click --> most common malware technique



macOS 15 - Sequoia

- Removed Right-click - open override
- Now users have to go to System Settings
- Important from malware point of view not really exploitation

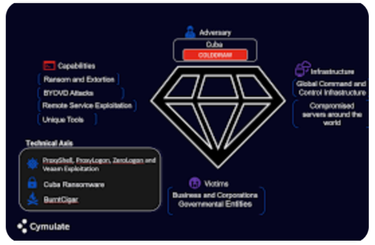
KEXTs

KEXTs

- Kernel EXTensions
- if loaded ==> kernel code exec ==> long time target for exploits
- Mac OS X 10.10 Yosemite (2014) ==> requires KEXT signing certificate
- macOS 10.13 High Sierra (2017) ==> SKEL (Secure Kernel Extension Loading) is introduced -> requires user approval

KEXT attacks - SKEL + BYOVD

- SKEL bypass by Patrick Wardle https://speakerdeck.com/patrickwardle/the-mouse-is-mightier-than-the-sword
- achieved via synthetic mouse events
- bypass SKEL -> load a vulnerable 3rd party driver
- exploit 3rd party driver to gain kernel code exec
  - Bring Your Own Vulnerable Driver (BYOVD) on the OS which shall not be named

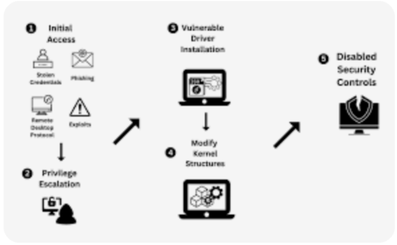


Cymulate

What are BYOVD Attacks? - Cymulate

Sangfor Technologies

What is BYOVD? – BYOVD Attacks in 2023



FourCore ATTACK

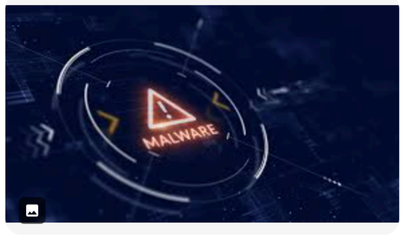
Exploit Party: Bring Your Own Vulnerable …

Sangfor Technologies

What is BYOVD? – BYOVD Att…

Cymulate

What are BYOVD Attacks? - …

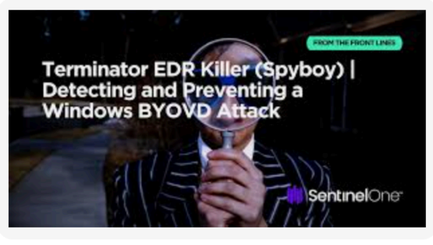


Barracuda Blog

Malware Brief: Crafty phishing, BYOVD an…

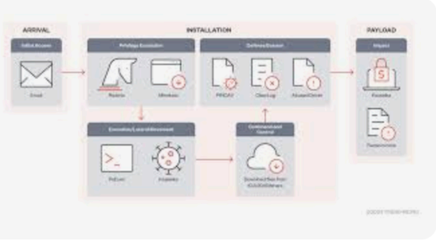
Cyberbit

BYOVD: Local privilege escalation via BioNTdrv…



SentinelOne

Terminator EDR Killer (Spyboy) | Prevent Wind…



Trend Micro

Kasseika Ransomware Deploys BYOVD Attacks …

ICT Security Magazine

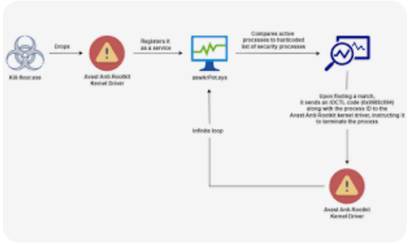
Bring Your Own Vulnerable Driver: l'ascesa in…

Global Business Outlook

BYOVD: The new threat for cybersecurity …

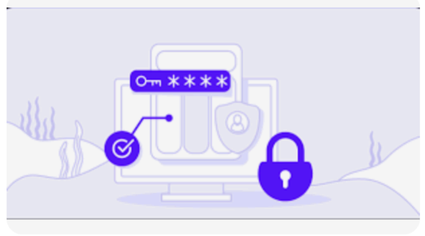
FourCore ATTACK

Exploit Party: Bring Your Own Vulnerable D…



The Hacker News

Researchers Uncover Malware Using BYO…

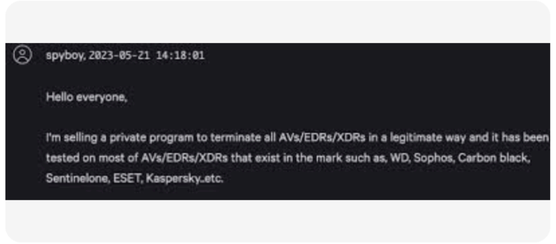


Cymulate

What are BYOVD Attacks? - Cymulate

LinkedIn

How Vulnerable Drivers Enable BYOVD A…



Sangfor Technologies

What is BYOVD? – BYOVD Attacks in 2023

KEXT attacks

- CVE-2020-9939 - Unsigned KEXT Load Vulnerability
  - part of an exploit chain used in pwn2own 2020 - https://github.com/sslab-gatech/pwn2own2020
  - start loading an Apple signed driver
  - swap driver after code signing verification
  - with use of symlinks
- CVE-2021-1779
  - same story, bypasses the patch

and then came Big Sur

- two major improvements:
  1. KEXT is staged into Auxiliary Kernel Extension Collection (SIP protected)
  2. Reboot is required => code signature is verified at load time
- an SKEL bypass could still work
- only 1 known bypass (Intel w/o T2 only) CVE-2022-46722 by Mickey Jin
  - https://objectivebythesea.org/v6/talks/OBTS_v6_mJin.pdf

Apple Silicon

- 3rd party KEXTs are disallowed *

**unless permitted in recovery mode*

Is Apple right?

- endless debate
- but!!!
  - major attack surface reduction
  - if attacker is in the kernel -> can do anything anyway

TCC

TCC

- Transparency, Consent and Control
- protects private data
- Mac OS X 10.8 Mountain Lion (2012): First release
- macOS 10.14 Mojave (2018): Major extension, lots of new categories
- ever growing categories since then

Private data everywhere

- grepping since 2019
- turns out private data is everywhere, not just where designed to be
- 30+ CVEs with private data leaks
- apps make copy of data and store it themselves

app data protection

- Apple closed the leaks 1 by 1
- eventually in Sonoma: protect every app's container
  - only applies to sandboxed apps
- closes most remaining and possible future leaks universally
- also solves downgrade attacks, if app is changed ==> alert

mount attacks - 2020 - 2023 - the golden era

- CVE-2020-9771 - TCC bypass via snapshot mounting
- CVE-2021-1784 - TCC bypass via mounting over com.apple.TCC
- CVE-2021-30782 - TCC bypass via AppTranslocation service
- CVE-2021-30947 - TCC bypass with Time Machine
- CVE-2022-22655 - TCC bypass admin configuration
- CVE-2022-22655 - TCC location services bypass
- CVE-2023-40425 - Enable private data in logs
- CVE-2023-42936 - Enable Private Data in Logs v2

mount protection

- now every new TCC protected location gets mount protection
- exceptions exists, but rare

Process Injection

When can we inject

- process is:
  - not hardened AND
  - not platform binary AND
  - not entitled
  - or has "get-task-allow" entitlement
- Mojave: most apps are injectable

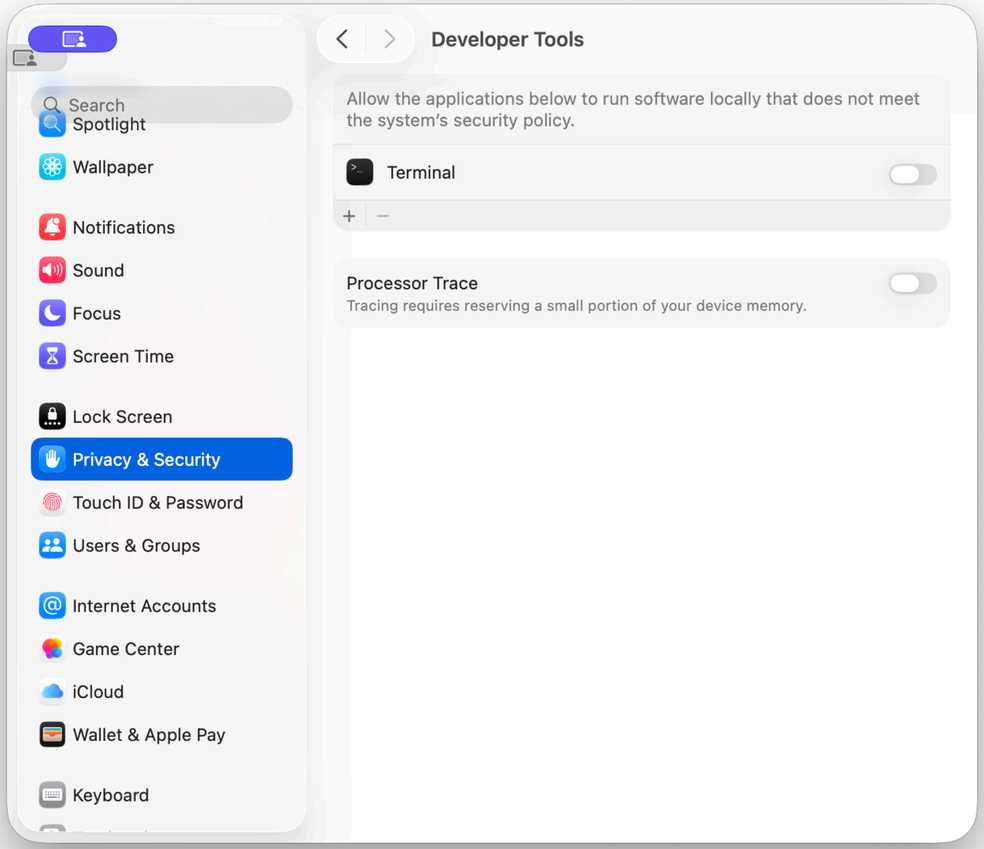
Hardened runtime

- Catalina: notarization kicks in
- soon hardened runtime becomes mandatory
- nowadays: non of the 3rd party processes are injectable *

**unless build with Electron...*

(?) Sequoia

- Developer Tools = NO ==> can't get the task port of anything (unless target signed with get-task-allow)



Launch Constraints

Let's review some exploits

TCC bypass with imagent.app

- Found by Adam Chester (@_xpn_)
- imagent.app with TCC and keychain related entitlements
- loads plugins from:
  - imagent.app/Contents/PlugIns
- code signing allows 3rd party plugins
- copy app to /tmp/ and load your plugin

```
<key>com.apple.private.tcc.allow.overridable</key>
<array>
        <string>kTCCServiceAddressBook</string>
</array>

<key>keychain-access-groups</key>
<array>
        <string>ichat</string>
        <string>apple</string>
        <string>appleaccount</string>
        <string>InternetAccounts</string>
        <string>IMCore</string>
</array>
```

TCC bypass using configd, "powerdir"

- Found by Jonathan Bar Or (@yo_yo_yo_jbo)
- configd has user update rights (can change HOME)
- -b allows loading an bundle (including non Apple)
- normally launched by launchd but we could start it via command line as well

```
[Key] com.apple.private.tcc.allow
[Value]
    [Array]
        [String] kTCCServiceSystemPolicySysAdminFiles
```

LC

Launch Constraints

- introduced in macOS Ventura (13)
- mitigates many logic vulnerabilities
- defines 3 constraints:
  - Self Constraints
  - Parent Constraints
  - Responsible Constraints

LC in Action

```
csaby@max /tmp % cp -r /System/Applications/FindMy.app .

csaby@max /tmp % open FindMy.app
The application cannot be opened for an unexpected reason, error=Error Domain=RBSRequestErrorDomain
Code=5 "Launch failed." UserInfo={NSLocalizedFailureReason=Launch failed.,
NSUnderlyingError=0x6000000032d0 {Error Domain=NSPOSIXErrorDomain Code=162 "Unknown error: 162"
UserInfo={NSLocalizedDescription=Launchd job spawn failed}}}
```

```
csaby@max /tmp % log stream | grep AMFI
2023-09-19 14:18:21.273482+0200 0x2e3486   Default     0x0                   0      0    kernel:
(AppleMobileFileIntegrity) AMFI: Launch Constraint Violation (enforcing), error info: c[1]p[1]m[1]e[2],
(Constraint not matched) launching proc[vc: 1 pid: 52468]: /private/tmp/FindMy.app/Contents/MacOS/
FindMy, launch type 0, failure proc [vc: 1 pid: 52468]: /private/tmp/FindMy.app/Contents/MacOS/FindMy
```

Launch Constraints Categories

LC Categories

- category = defines a set of launch constraints
- Ventura - 7 categories - documented by Linus Henze
- Sonoma - 18 categories - documented by Csaba Fitzl
- assigns each binary in the trust cache to a category

LC Category examples

```
Category 1:

    Self Constraint: (on-authorized-authapfs-volume || on-system-volume) && launch-type == 1 &&
validation-category == 1

    Parent Constraint: is-init-proc
```

- *on-authorized-authapfs-volume || on-system-volume* - System or Cryptex
- *launch-type == 1* - system service
- *validation-category == 1* - must present in the trust cache
- *is-init-proc* - launchd

```
/usr/libexec/routined
/usr/libexec/nehelper
/usr/libexec/remoted
/usr/libexec/seld
/usr/libexec/logd
/usr/libexec/thermalmonitord
```

LC Category examples

```
Category 2:

     Self Constraint: on-authorized-authapfs-volume || on-system-volume
```

- *on-authorized-authapfs-volume || on-system-volume* - System or Cryptex
- less restrictive

```
/usr/bin/brctl
/usr/bin/bputil
/usr/bin/bison
/usr/bin/bioutil
/usr/bin/binhex
/usr/bin/bc
/usr/bin/batch
```

attack mitigation

LC attack mitigation

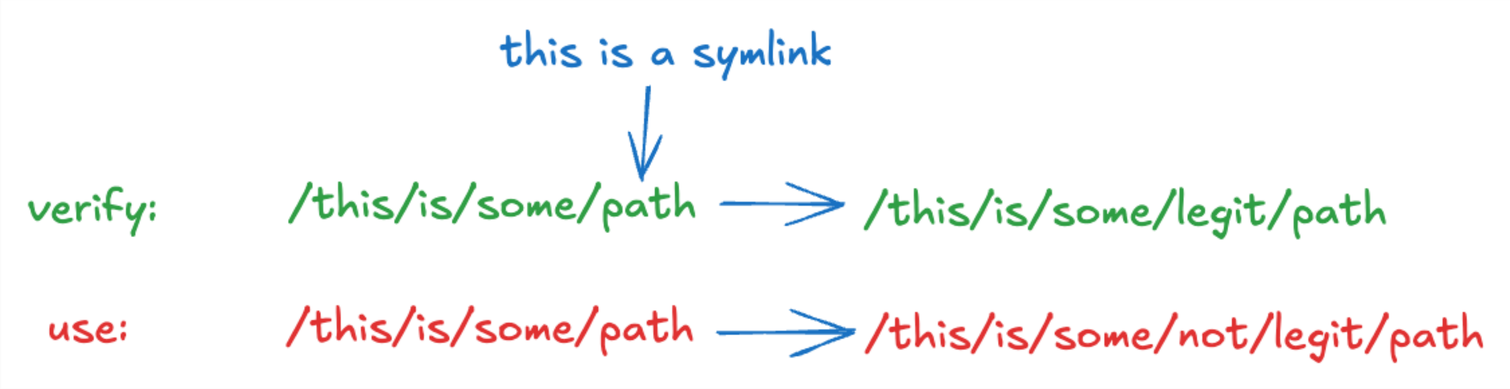
- imagent.app
  - (on-authorized-authapfs-volume || on-system-volume)
  - wouldn't be able to start a copy

- configd
  - Parent Constraint: is-init-proc + system service
  - wouldn't be able to start from command line

File Operations

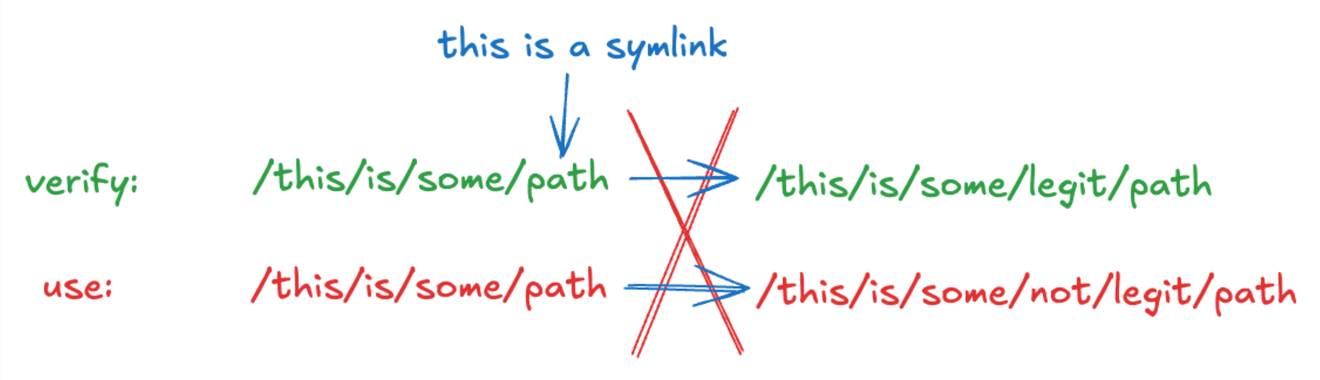
Symlink Attacks

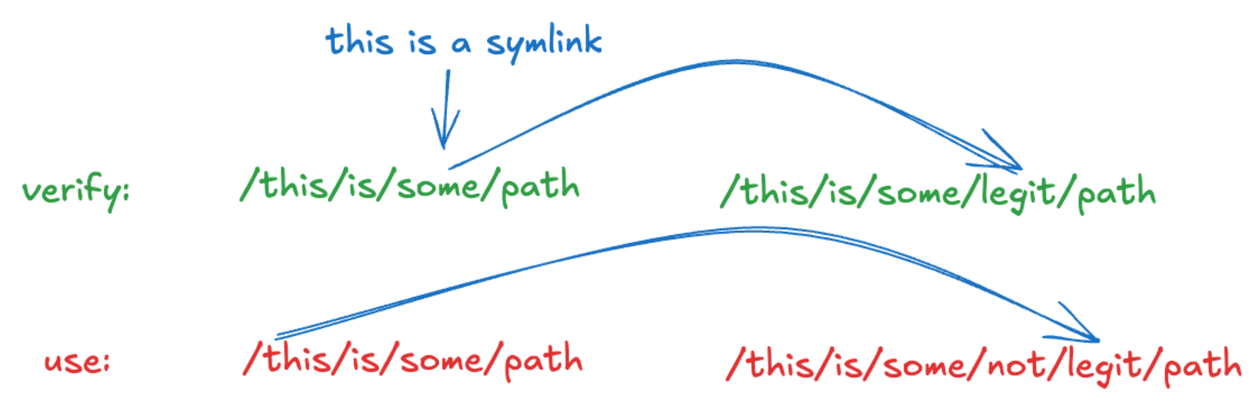
- redirect file operations with a symlink
- common TOCTOU attack (time of check time of use)



O_NOFOLLOW

- don't follow symlinks
- problem: only checks last path component





O_NOFOLLOW_ANY

- available since 2022
- none of the path components can be symlink
- getting more and more widespread
- mitigates most of the symlink attacks if used properly

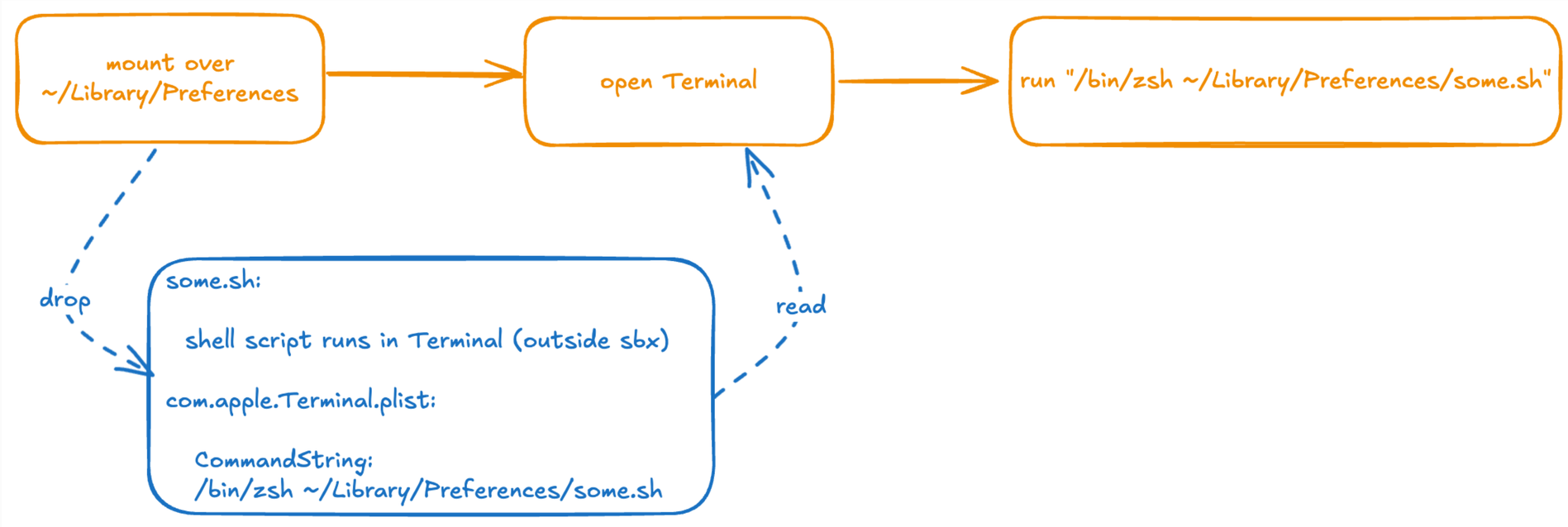
Closing weaponization paths

Weap... WHAT?

- weaponization ~ turn an exploit into useful code execution
  - e.g. you can:
  - mount anywhere
  - drop a file
  - modify a file permission
  - create a directory with user's permission
  - etc...
  - ==> turn them to code exec as root, sb escape, tcc bypass, etc...

Trick 1

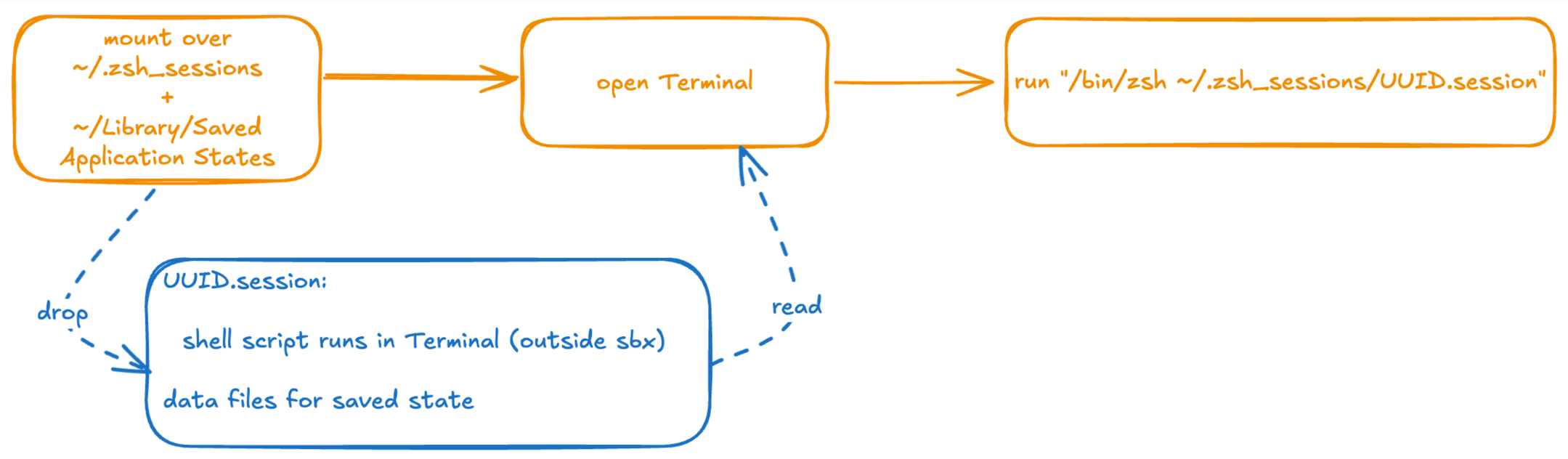
- Can mount anywhere from Sandbox



- Closed: macOS Sequoia (Preferences is TCC protected)

Trick 2

- Can mount anywhere from Sandbox



- Closed: macOS Sequoia/Tahoe (Saved State is TCC protected)

Trick 3

- Can bypass SIP
- modify: /Library/Apple/Library/Bundles/TCC_Compatibility.bundle/Contents/Resources/AllowApplicationsList.plist (=TCC.db)
- Closed: macOS Sequoia (no longer supported, file is not available)

Trick 4

- Can mount or drop file as root
- Use periodic scripts
- Closed: macOS Big Sur / Monterey (TCC protected)

Trick 5

- Can drop any file as root but with user ownership
- use /Library/LaunchDaemons
- Closed ~ Big Sur, file ownership must be root

Trick 6

- Can drop any file as root but with user ownership
- use /etc/pam.d
- Closed ~ Big Sur (pam.d is TCC protected)

conclusion

conclusion

- Apple is raising the bar continuously
- existing features gets improved
- lots of weaponization paths are closed
- logic exploitation gets harder and harder

kandji

Csaba Fitzl
Twitter: @theevilbit

Resources

- flaticon.com - Freepik, rsetiawan