ನಮಸ್ಕಾರ

Is This App Safe ?

Guide to macOS Binary Verification

#binary-triage

Vishal Chand
Researcher (BharatGen), IIT Bombay

$whoami?

- Fusion of LLMs with cybersecurity
- Digital Forensics and Incident Response trainer @Critical Information Infrastructure Security Exericse 2025 which was conducted by NCIIPC, A unit of NTRO
- VERY passionate about National security

Agenda ?

Help you confidently answer "Is this app safe to run?"

To inspire you to explore macOS security.

Do Macs even get malware?

Mac doesn’t get PC viruses. A Mac isn’t susceptible to the thousands of viruses plaguing Windows-based computers. That’s thanks to built-in defenses in Mac OS X that keep you safe without any work on your part.

Cross platform malware: NimDoor, BeaverTail, JaskaGO

macOS malware history

1982	Elk Cloner	First Apple virus, spread via floppy disks
1987–89	nVIR, HyperCard	Early Mac OS viruses, humorous messages
1998	AutoStart Worms	Spread via CD-ROM AutoPlay, data loss
2004	MP3Concept, Renepo	Early Mac OS X trojans, prompted security boost
2006	Leap-A	Spread via iChat, limited effect
2007	RSPlug	DNS changer trojan
2009	MacSweeper, Genieo	Rogueware, adware
2011	MacDefender	Fake alerts, credit card theft
2012	Flashback	Java exploit, mass infection
2014	iWorm	Backdoor, 17,000+ infections
2016	KeRanger	First Mac ransomware

macOS malware history

2017	APT28/XAgent	Modular espionage backdoor
2018–2023	Lazarus, RustBucket	State-sponsored, cross-platform, supply chain

NOTE: 22 New Mac Malware Families Seen in 2024 which includes stealers, backdoors, downloaders and ransomware.

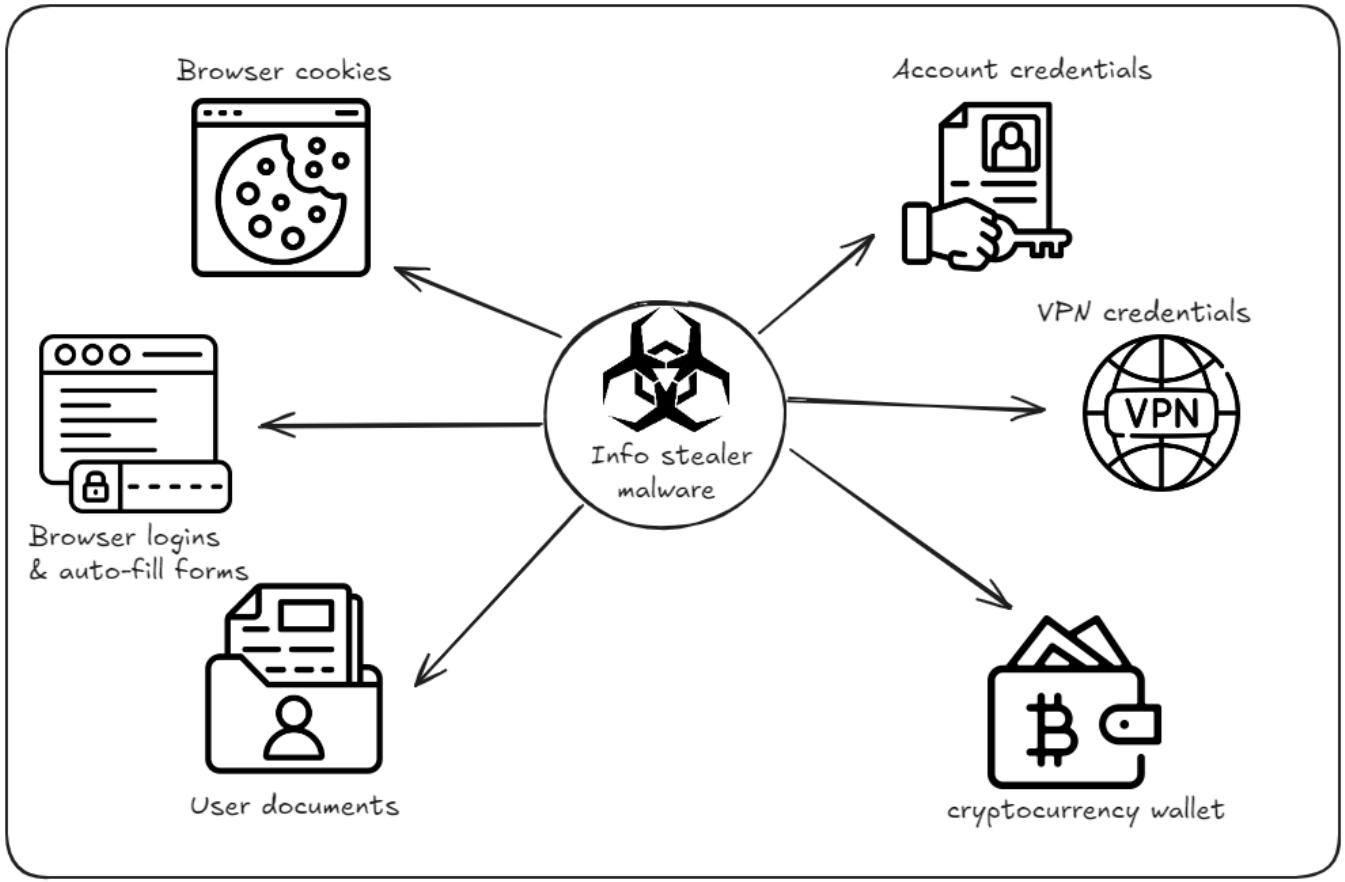
Why it matters ?

- Mac user base is large & growing
- Approximately **100.4 million** people use Macs in 2024.
- Apple devices now seen “**across the Fortune Top 500**”
- Mac Malware is surging
- Mac will become the dominant enterprise endpoint by 2030." -Jamf
- Use of Zero-days , Sophisticated targeting, Advanced stealth techniques (Lazarus APT)

Mac Malware: It's Insidious!!!

- Most common type macOS malware in 2023: Info stealer
- Last year (2024) … again it was infostealer !
- Infostealers targeting macOS jumped by 101% last year
- (recent) macOS Malware :

- Cloudchat
- AMOS
- Poseidon
- Cthulhu
- BeaverTail
- Pystealer
- Banshee
- NotLockBit
- SpectralBlur
- HZ Rat
- RiddenRisk
- RustyAttr
- DPRK downloader
- ToDoSwift
- InletDrift

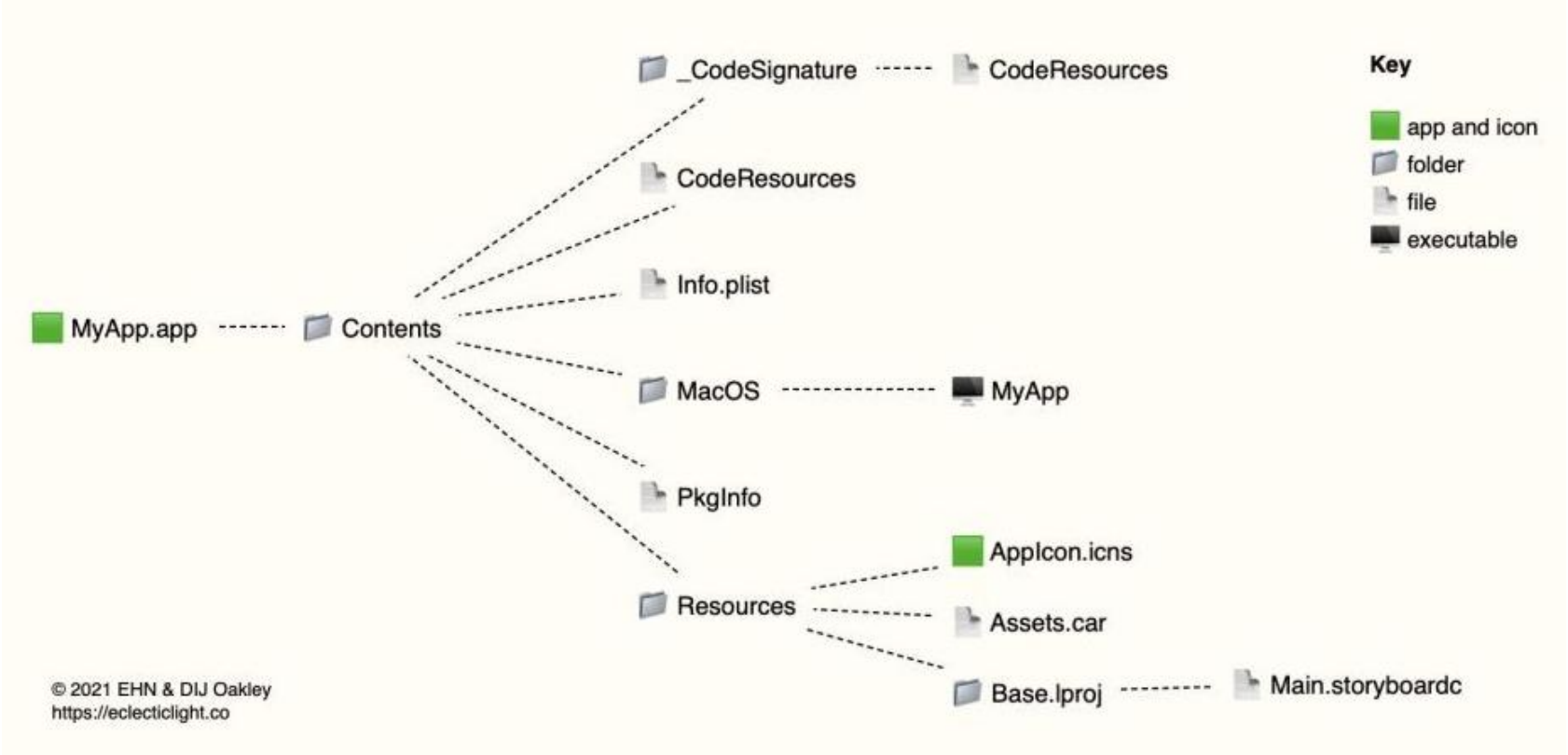

Browser cookies
Account credentials
VPN credentials
VPN
Info stealer malware
Browser logins & auto-fill forms
User documents
cryptocurrency wallet

Mac Malware: It's Insidious!!!

- Most common type macOS malware in 2023: Info stealer
- Last year (2024) … again it was infostealer !
- Infostealers targeting macOS jumped by 101% last year
- (recent) macOS Malware :

- Cloudchat
- AMOS
- Poseidon
- Cthulhu
- BeaverTail
- Pystealer
- Banshee
- NotLockBit
- SpectralBlur
- HZ Rat
- RiddenRisk
- RustyAttr
- DPRK downloader
- ToDoSwift
- InletDrift

Structure of a basic app

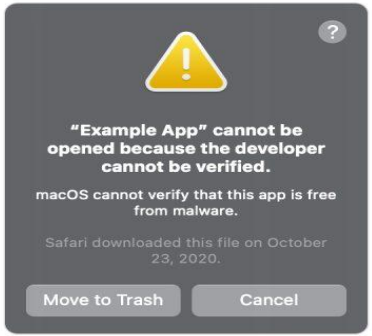


Three layers of defence in macOS

1. Prevent launch or execution of malware: **App Store**, or **Gatekeeper** combined with **Notarisation**

2. Block malware from running on customer systems: Gatekeeper, Notarisation and **XProtect**

3. Remediate malware that has executed: XProtect



https://support.apple.com/en-in/guide/security/sec469d47bd8/web

The App store (first line of defence)

Protecting App Store Users: App Review in 2022

Nearly 1.7 million app submissions rejected

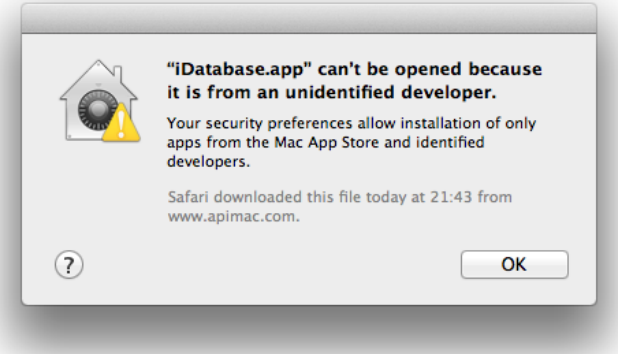
Nearly 400,000
app submissions rejected for privacy violations

Over 153,000
app submissions rejected for spam, copycats, or misleading users

Nearly 29,000
app submissions rejected for containing hidden or undocumented features

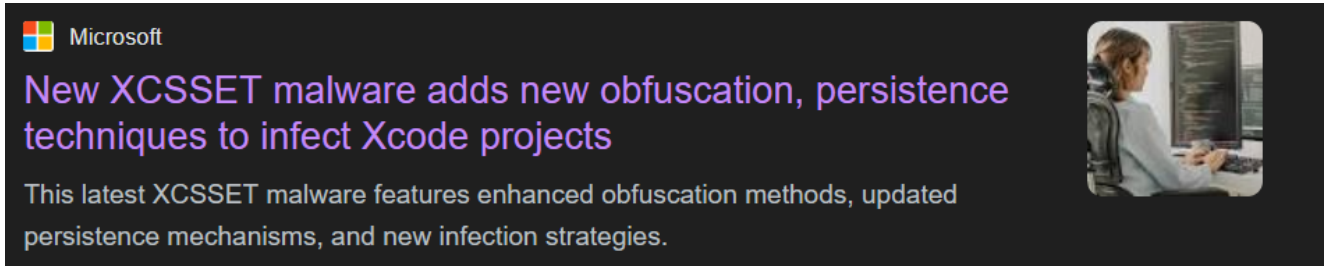
macOS Gatekeeper

Gatekeeper ensures that only trusted software can run on a Mac. It verifies the app's trustworthiness the first time it's opened based on two factors: code signing and notarization.



Code signing:

- Code signing ensures the app is from a known developer.
- Code signing guarantees app comes from a trusted source
- Does not guarantee that the code itself is safe.



Apple XProtect

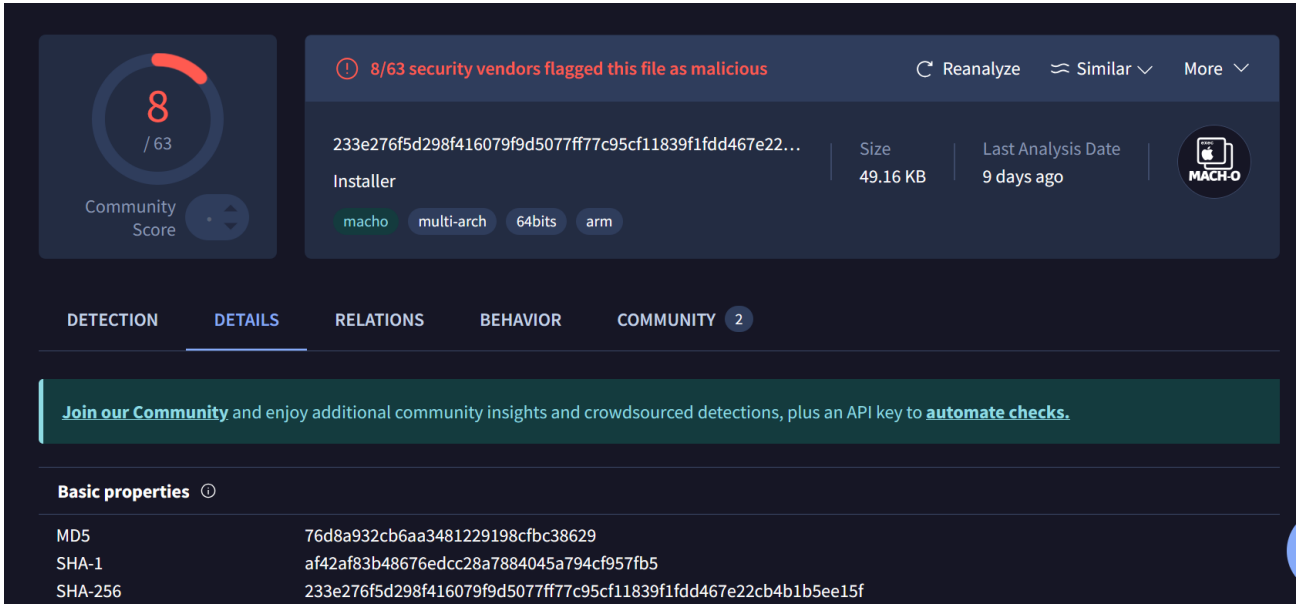
- mac OS includes built-in antivirus technology called *XProtect* for the signature-based detection and removal of malware.
- Uses YARA (for signature-based detection of malware)

Banshee Malware : How it outsmarted Xprotect by stealing it's magic

- Banshee's author "stole" the **string encryption algorithm** from Apple's MacOS XProtect antivirus engine

Hash

```
% md5 0x16.app/Contents/MacOS/usrnode
  MD5 (usrnode) = 76d8a932cb6aa3481229198cfbc38629
```



Check the code signing information

Ensure the binary is signed by a known and trusted Apple Developer ID.

- macOS uses code signing as a core security feature. It cryptographically validates:

- The origin of the app (Developer ID)
- The integrity of the binary (no modifications since signing)
- The entitlements and resources bound to the binary

```
% codesign -dvv 0x16.app/Contents/MacOS/usrnode
Executable=0x16.app/Contents/MacOS/usrnode
Identifier=com.alis.tre
Format=app bundle with Mach-O thin (x86_64)
Authority=(unavailable)
TeamIdentifier=95RKE2AA8F
...
```

Red flags to look for :

1. code object is not signed at all
2. signature does not match

```
% codesign -dvv Calculator.app
Executable=Calculator.app/Contents/MacOS/Calculator
Identifier=com.apple.calculator
Format=app bundle with Mach-O universal (x86_64 arm64e)
Authority=Software Signing
Authority=Apple Code Signing Certification Authority
Authority=Apple Root CA
...
```

Notarization

- Developers submit their apps to Apple for notarization.
- Apple notarizes apps only after checking that they are:

- from a recognized developer (meaning it’s signed with a Developer ID certificate);
- free of malicious content; and
- free of code-signing issues.

- If an app is later found to be malicious, Apple can also revoke such tickets

Notarization and Staple Check

What is a Stapled Notarization Ticket in macOS?

In macOS, stapling ref **app to be**

verified offline by Gat

What to Look For: sou

Apple Approved Malware

Apple Approved Malware

Campaign originating from homebrew.sh, leveraged adware payloads were actually fully notarized!

```
$ spctl -a -vvv -t install /Volumes/Install/Installer.app
/Volumes/Install/Installer.app: accepted
source=Notarized Developer ID
origin=Developer ID Application: Morgan Sipe (4X5KZ42L4B)

$ spctl -a -vvv -t install /Users/patrick/Downloads/Player.pkg
/Users/patrick/Downloads/Player.pkg: accepted
source=Notarized Developer ID
origin=Developer ID Installer: Darien Watkins (NC43XU5Z95)
```

https://objective-see.org/blog/blog_0x4E.html

Check the Entire App Bundle for Suspicious Attributes

- macOS attaches metadata to files using extended attributes (xattrs).
- These are key-value pairs stored outside the file's contents,

Why checking Entire App Bundle matters?

- Determine how an app entered the system (e.g., from Safari, Mail, USB).
- Detect whether it was quarantined by Gatekeeper.

```
find /path/to/App.app -exec xattr -l {} +
```

Red flags: missing quarantine, shady WhereFroms

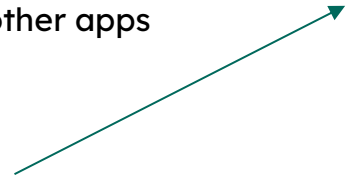
```
com.apple.quarantine: 0083;5f8c2b4d;Safari;E5B6D9A7-0000-0000-AFAF-111122223333
com.apple.metadata:kMDItemWhereFroms:
https://good-cat.com/download
```

Check the entitlements of the app Bundle or Executable

Entitlements are key-value pairs that grant to executable permission to use a service or technology.

High-Risk Entitlements

- com.apple.security.automation.apple-events - Control other apps
- com.apple.security.network.client - Network access
- com.apple.security.device.camera - Camera access



This will trigger TCC (Transparency, Consent and Control) prompt.

```
codesign -d --entitlements :- /path/to/App.app/Contents/MacOS/executable
```

Check the entitlements of the app Bundle or Executable (Contd...)

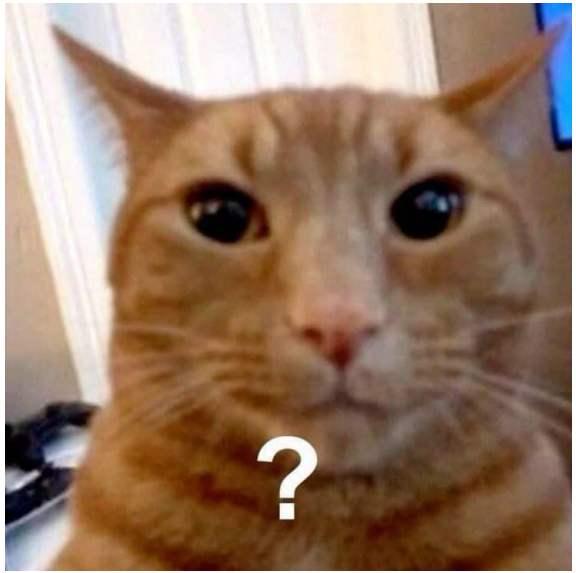
Alternatively, we can use jtool : **jtool2 -ent ~/mybaby_malware.app/Contents/MacOS/mybaby_malware**

```
<?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.security.app-sandbox</key>
    <true/>
    <key>com.apple.security.cs.allow-dyld-environment-variables</key>
    <true/>
    <key>com.apple.security.cs.disable-library-validation</key>
    <true/>
    <key>com.apple.security.files.user-selected.read-only</key>
    <true/>
    <key>com.apple.security.network.client</key>
    <true/>
    <key>com.apple.security.network.server</key>
    <true/>
    <key>com.apple.security.files.user-selected.read-write</key>
    <true/>
    <key>com.apple.security.device.camera</key>
    <true/>
    <key>com.apple.security.device.audio-input</key>
    <true/>
    <key>com.apple.security.cloudkit</key>
    <true/>
    <key>com.apple.security.print</key>
    <true/>
    <key>com.apple.security.automation.apple-events</key>
    <true/>
    <key>com.apple.security.device.usb</key>
    <true/>
</dict>
</plist>
```

Check the entitlements of the app Bundle or Executable (Contd...)

Questions to Ask:

- Does a simple calculator need camera access?
- Why does this app need network permissions?
- Are the entitlements proportional to functionality?



Extracting Symbols

- Binary's symbols contain the names of the binary's functions or methods and those of the API it imports.

- Reveals the file's capapabilites

```
% nm malware_sample3
...
"+[Exec doShellInCmd:]",
"-[ShellClassObject startPty]",
"-[MethodClass getIPAddress]",
"-[MouseClassObject PostMouseEvent::::]",
"-[KeychainClassObject getPasswordFromSecKeychainItemRef:]"
...
```

Endpoint Security Framework (ESF)

ESF = Apple's Powerful Monitoring API

What is ESF?

- Kernel-level monitoring of system events
- Real-time notifications for file, process, network events
- Replacement for deprecated kauth framework
- Required for enterprise security tools

To use ESF, your tool must: **BE SIGNED WITH A SPECIAL ENTITLEMENT**

NOTE : You can't invoke esf from shell like codesign, you must build tools using C/Swift and entitlements **(com.apple.developer.endpoint-security.client).**

Endpoint Security Framework (ESF)

ESLogger - Built-in ESF Tool

eslogger is a command-line tool built on top of Apple's Endpoint Security Framework (ESF) that logs real-time security-relevant events on macOS systems such as:

- Binary execution (exec)
- File modifications (unlink, rename)
- Process forking
- Privilege escalations
- File writes, opens, and memory mappings

Endpoint Security Framework (ESF)

ESLogger - Built-in ESF Tool

- Subscribes to low-level **kernel event hooks** using a **system extension** or privileged entitlement.
- Outputs structured JSON logs of each event.

```
1 sudo eslogger exec | jq .
============================================================
{
"event_type": "ES_EVENT_TYPE_AUTH_EXEC", 2
"process": {
"executable": "/usr/bin/suspicious_app", 3
"pid": 1234,
"signing_id": "com.unknown.suspicious" 4
}
}
============================================================ 5
sudo eslogger exec | jq 'select(.code_signature.signed == false)'
============================================================
"code_signature": {
"signed": false
}
6
```

Endpoint Security Framework (ESF)

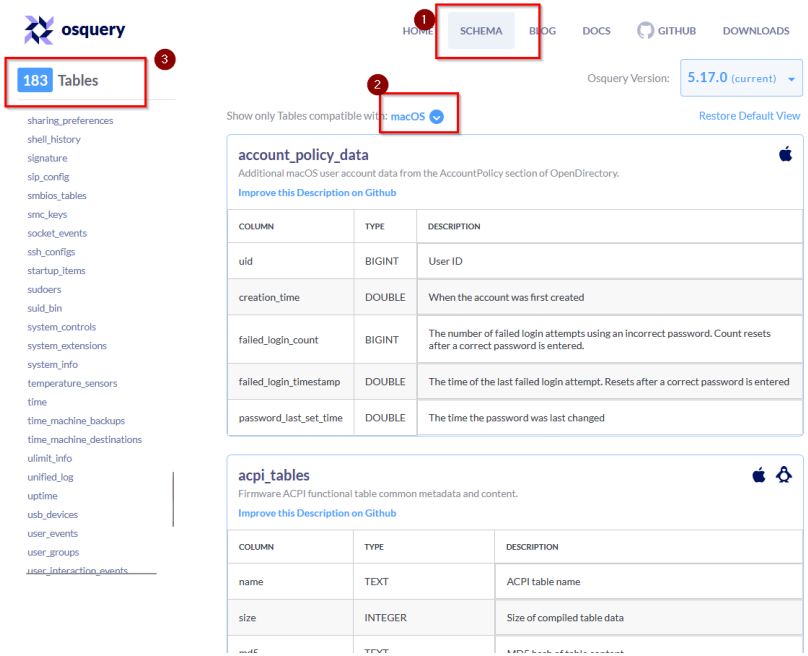
Key Event Types to Monitor:

- **ES_EVENT_TYPE_AUTH_EXEC** - Process execution
- **ES_EVENT_TYPE_AUTH_OPEN** - File opens
- **ES_EVENT_TYPE_NOTIFY_WRITE** - File modifications
- **ES_EVENT_TYPE_AUTH_KEXTLOAD** - Kernel extension loads
- **ES_EVENT_TYPE_NOTIFY_PROC_EXIT** - Process termination

Use cases : Malware analysis , Behavior analysis, Threat telemetry, Offensive testing & DFIR

Osquery

- Osquery is an open-source agent created by Facebook in 2014
- Converts the OS into a relational database
- Using basic SQL commands, you can ask questions about devices, such as servers, Docker containers, and computers running Linux, macOS, or Windows
- The extensive schema helps with a variety of use cases including vulnerability detection, compliance monitoring, incident investigation and more.
- Osquery interfaces with the kernel (e.g., openbsm, kaudit, etw) to capture kernel behavioral activity/events (e.g., processes launched, socket connections, file changes). This is done via event tables.



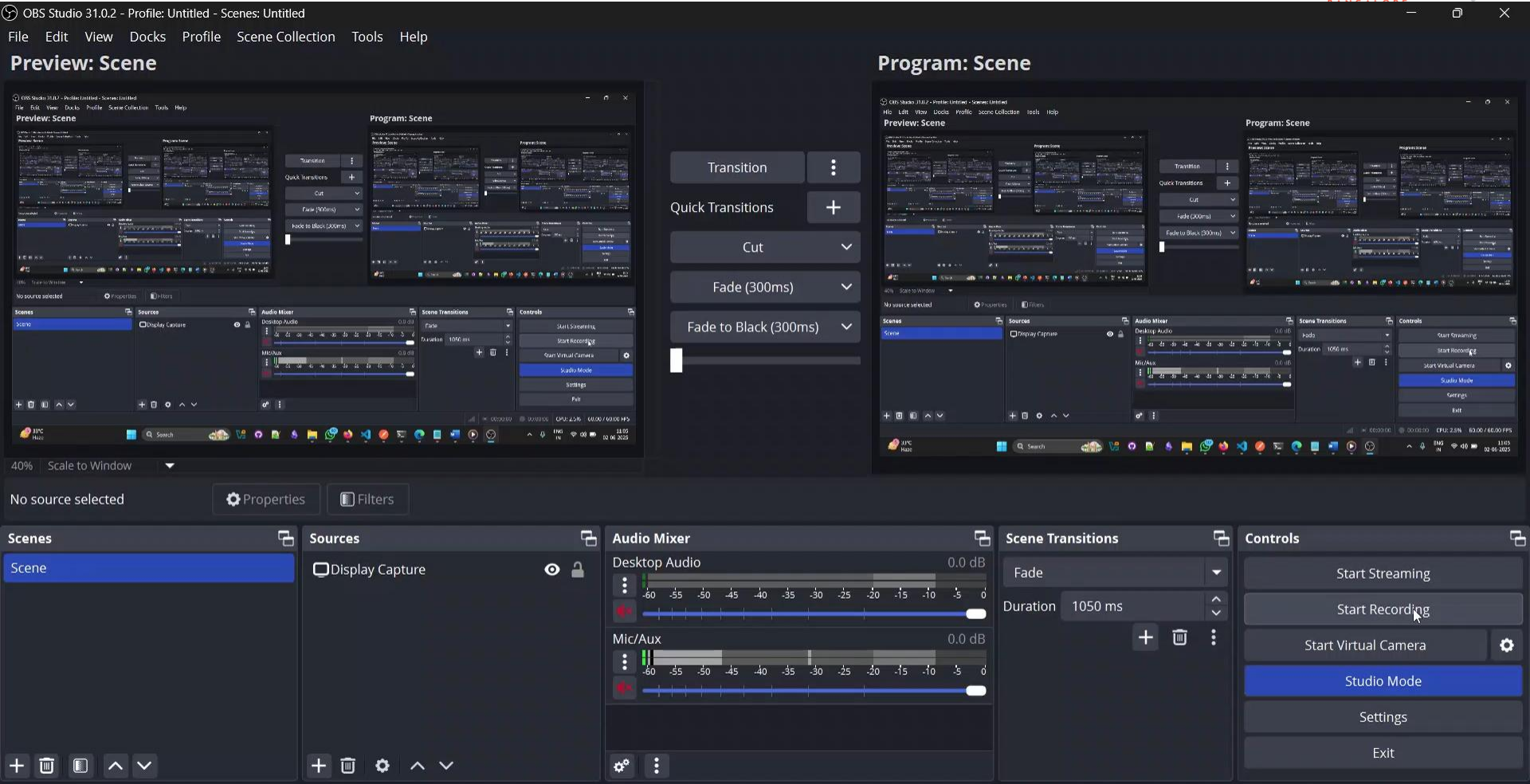
Analysis using osquery

```
.osquery> .mode line
.osquery> select * from process_events;
        pid = 2549
       path = /Users/vishal/Downloads/OSX.Dummy/script
       mode = 0100744
    cmdline = ./script
        cwd =
       auid = 501
        uid = 0
       euid = 0
        gid = 0
       egid = 0
  owner_uid = 501
  owner_gid = 20
      atime =
      mtime =
      ctime =
      btime =
     parent = 2546
```

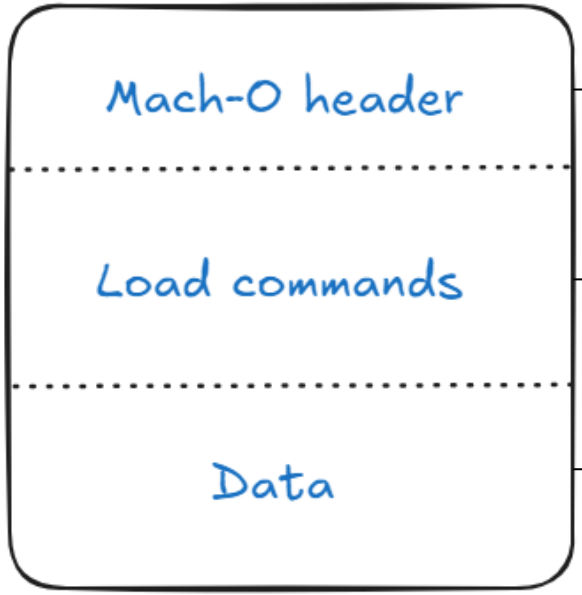
1

2

Osquery + Large Language Model (LLMs) -> NL2SQL



Layout of a Mach-O-Binary



CPU architecture , Type of Mach-O format

Tell dynamic loader (dyld) how to load & link the binary in memory.

Largely consist of actual binary code

Check load commands

Load commands in a Mach-O binary (macOS executable format) are instructions for the dynamic linker (dyld) about how to prepare and load the binary.

- LC_LOAD_DYLIB (Loads a shared library at runtime)
- LC_MAIN (Entry point of binary)
- LC_PATH (Runtime library search paths)
- LC_CODE_SIGNATURE (Signature info location)
- LC_SEGMENT /LC_SEGMENT_64 (Defines memory segments)

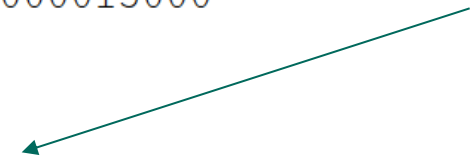
```
otool -l /path/to/App.app/Contents/MacOS/executable
```

Check load commands

```
% otool -lv 0x16.app/Contents/MacOS/usrnode
...
Load command 1
cmd LC_SEGMENT_64
cmdsize 952
segname __TEXT
vmaddr 0x0000000100000000
vmsize 0x0000000000013000
fileoff 0
filesize 77824
maxprot rwx
initprot r-x
nsects 11
flags (none)
```

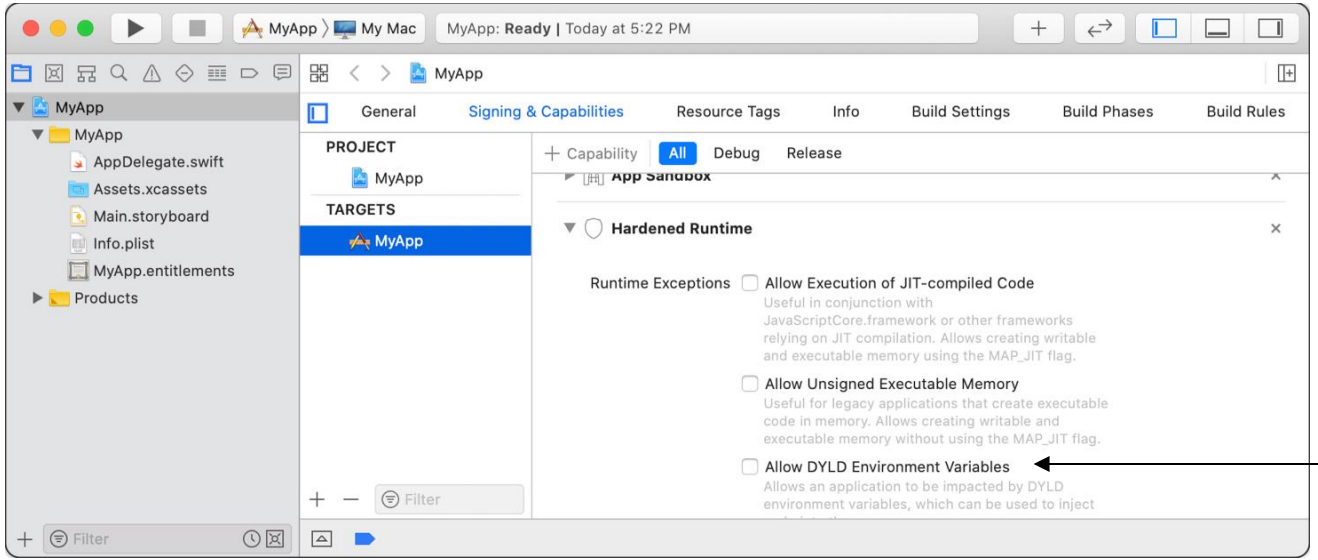
1

2



Check for dylibs used

- Dynamic libraries are modules containing executable code that a process can load and execute.
- DYLD_* Environment Variable : DYLD_INSERT_LIBARIES & DYLD_FRAMEWORK_PATH
- Apple security : **Hardened Runtime (Required for macOS app to be notarized)**



Check for dylibs used

- Dynamic libraries are modules containing executable code that a process can load and execute.
- DYLD_* Environment Variable : DYLD_INSERT_LIBARIES & DYLD_FRAMEWORK_PATH
- Apple security : **Hardened Runtime (Required for macOS app to be notarized)**
- Hardened Runtime exceptions**: com.apple.security.cs.allow-dyld-environment-variables** (⚠ can allow malicious dylibs to load)
- **Dylib Proxying :** Replaces a legitimate dylib dependency (e.g., via path hijacking, bundle injection) with a malicious dylib to hijack execution or gain persistence.

Check for dylibs used

```
% otool -L iTerm.app/Contents/MacOS/iTerm2
/usr/lib/libaprutil-1.0.dylib
...
@executable_path/../Frameworks/libcrypto.2.dylib
```

RED FLAGS ? WHAT TO LOOK FOR ? ?

- Linking to libcrypto.dylib, libssl.dylib, etc., from non-system locations. (e.g., /Users/, /tmp/, /private/)
- .dylib with names like libhelper.dylib, libinject.dylib, etc
- Custom libraries from temp/user paths
- Use of @rpath or @loader_path to load malicious libraries at runtime.

Check Imports and Exports

Imports: External functions the binary depends on (e.g., malloc, printf, execve)

Exports: Functions the binary exposes, like APIs or plugin hooks

Malware often imports functions like ptrace, mmap, or system() to perform anti-analysis, memory injection, or command execution.

What to check? Exports (T-prefixed)

- Custom exported APIs (_payload_handler, _hooked_send) : May indicate malware modules
- Obfuscated names like _x1a2b3c : Common in packed/encrypted payloads

```
% nm malware_sample1
000000010000a550 T _payload_handler
000000010000b120 T _react_exec
```

react_* functions contains logic to execute command from C2

Check Imports and Exports

What to check? Imports (U-prefixed)

- _ptrace : Used to detect/debug analysts
- _fork : Used to spawn child processes- common in persistence/backdoors
- _dlopen : Dynamically loads .dylibs can be used for sideloading
- _namp, _mprotect : Memory manipulation, often used in shellcode injection
- _system, _popen, _execve : Executes shell commands or scripts

```
% nm WindTail/0x16.app/Contents/MacOS/usrnode
 ...
 U _LSSharedFileListCreate
 U _LSSharedFileListInsertItemURL
 U _NSApplicationMain
 ...
 U _NSHomeDirectory
 U _NSUserName
```

Check Logging (ES, Unified, Crash Logs, Sysdiagnose)

Endpoint Security (ES) Logs

Requires writing a custom agent using the Endpoint Security Framework (ESF) in Swift/C.

- Captures low-level events like: exec, fork, mmap, unlink, open, socket, setuid, etc.
- Real-time JSON logs
- Typically used by EDR solutions

Requires Apple entitlement: com.apple.developer.endpoint-security.client

Check Logging (ES, Unified, Crash Logs, Sysdiagnose)

Unified Logging (Console)

macOS uses a centralized logging system. You can stream logs in real-time using:

Use this to detect: Access to restricted files , Entitlement failures, Sandbox violations, Networking activity

```
$ log stream --predicate 'subsystem contains "com.apple.security" or process == "amfid" or process == "tccd"'

Timestamp                       Thread     Type        Activity     PID    TTL
2025-07-05 14:24:15.123456-0700  0x2c3d4e   Error       0x0          0      -    kernel: (Security) CODE SIGNING: process 9876[untrusted_app]: missing or invalid entitlement: com.apple.se
2025-07-05 14:24:15.234567-0700  0x2c3d4e   Error       0x0          52     -    tccd: (TCCDProcess) Prompting policy for service kTCCServiceSystemPolicyDesktopFolder for requesting PID:
2025-07-05 14:24:16.345678-0700  0x2c3d4e   Error       0x0          0      -    kernel: (Security) CODE SIGNING: process 4321[suspicious_tool]: rejecting invalid signature: code signatur
2025-07-05 14:24:17.456789-0700  0x2c3d4e   Error       0x0          78     -    amfid: (Security) signature evaluation failed: code signature not valid for running pid 4321
2025-07-05 14:27:30.123456-0700  0x5a6b7c   Error       0x0          89     -    authd: (Authorization) Authorization denied for user 'vishal' trying to access privileged operation
```

Check Logging (ES, Unified, Crash Logs, Sysdiagnose)

Crash Logs

- macOS stores crash reports for each app in: ls **-lt ~/Library/Logs/DiagnosticReports**
- Each crash log contains:Stack trace ,Exception type (e.g., EXC_BAD_ACCESS, SIGABRT) ,Binary UUIDs and versions , Possible malware indicators (e.g., memory corruption)

Run dynamically & Observe

dtruss – Trace System Calls

sudo dtruss -p $(pgrep AppName)

- Purpose: Monitors low-level system calls (like open, read, write, fork, etc.)
- Use case: See if the app tries to access unusual files, run hidden processes, or tamper with system internals.
- E.g : Malware often checks system integrity or injects into other processes—dtruss will show ptrace() or mach_vm_* calls.

fs_usage – Monitor File System Activity

sudo fs_usage -w -f filesys

- Purpose: Real-time view of file system activity (reads, writes, creations, deletions).
- Track whether the app drops new files, especially in suspicious locations like ~/Library/LaunchAgents.

Run dynamically & Observe

```
fs_usage -w -f filesystem
  access     (___F)    com.apple.audio.driver.app/Contents/MacOS/conx.wol
  open       F=3       (R_____)  com.apple.audio.driver.app/Contents/MacOS/conx.wol
  flock      F=3
  read       F=3       B=0x92
  close      F=3
```

```
% cat com.apple.audio.driver.app/Contents/MacOS/conx.wol
{
   "PO": 80,
   "HO": "45.77.49.118",
   "MU": "CRHHrHQuw JOlybkgerD",
   "VN": "Mac_Vic",
   "LN": "adobe_logs.log",
   "KL": true,
   "RN": true,
   "PN": "com.apple.audio.driver"
}
```

Run dynamically & Observe

opensnoop – Track File Opens

sudo opensnoop -n AppName

- Purpose: Tells you every file the app is opening.
- Use case: Confirm whether the app is reading things it shouldn't—like private keys, cookies, etc.
- Real-life example: A malicious app reading ~/Documents/Passwords.txt would be caught.

lsof – List Open Files for a Process

sudo lsof -p $(pgrep AppName)

- Purpose: Lists all files (including sockets and pipes) currently opened by the app.
- Use case: See if the app is making network connections (sockets), or keeping files locked.
- Real-life example: Malware connecting to a C2 server via a raw TCP socket.

Run dynamically & Observe

```
% lsof -i TCP
COMMAND     PID  USER   TYPE      NAME
quicklookd  733  user   IPv4 TCP  192.168.0.128:49291->185.49.69.210:http (SYN
                                                                          SENT)
%
% ps -p 733
PID  TTY  CMD
733  ??   ~/Library/Dropbox/quicklookd
```

THANK YOU!

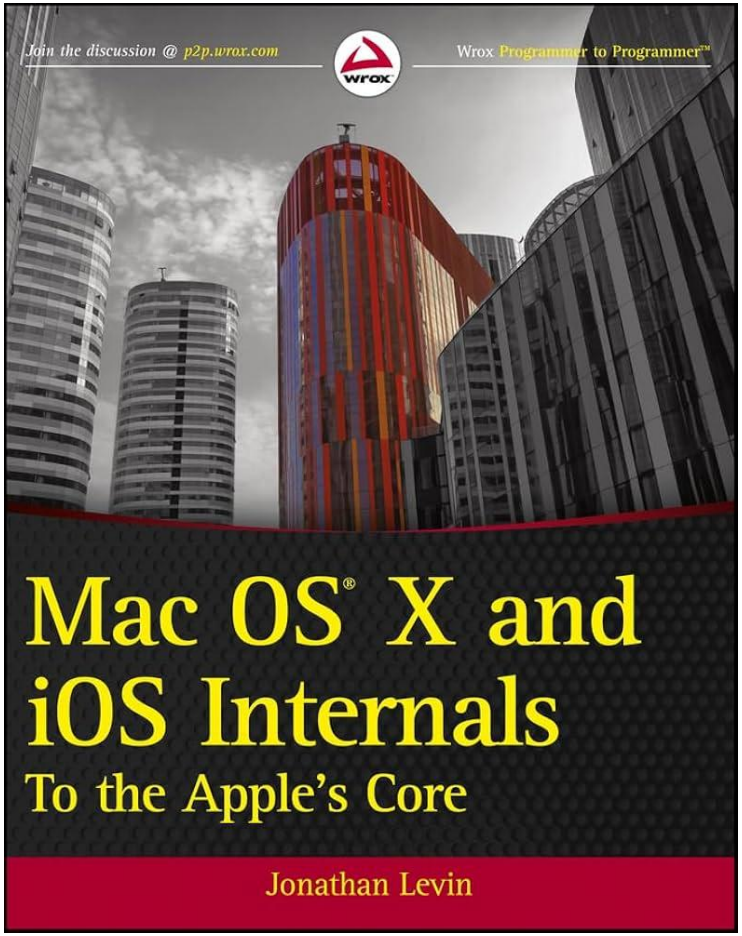
Adhokshaj Mishra, Staff Detection Engineer , SentinelOne

Bhargav Rathod, Security Analyst, Salesforce

References

1. https://www.securityweek.com/22-new-mac-malware-families-seen-in-2024/
2. https://developer.apple.com/documentation/security/code-signing-services
3. https://www.spyhunter.com/shm/macos-stats/
4. https://moonlock.com/macos-security-trends
5. https://objective-see.org/blog/blog_0x7D.html
6. https://support.apple.com/en-in/guide/security/sec469d47bd8/web)
7. https://www.microsoft.com/en-us/security/blog/2025/03/11/new-xcsset-malware-adds-new-obfuscation-persistence-techniques-to-infect-xcode-projects/
8. https://objective-see.org/blog/blog_0x4E.html
9. https://www.osquery.io/schema/5.17.0/
10. https://developer.apple.com/documentation/security/hardened-runtime
11. https://objective-see.org/index.html


CAN WE TALK OFF THE RECORD AS FRIENDS?


Join the discussion @ p2p.wrox.com
wrox
Wrox Programmer to Programmer™
Mac OS® X and iOS Internals
To the Apple's Core
Jonathan Levin

Objective-See
a non-profit 501(c)(3) foundation.

tools blog Support Us!

Objective-See's Blog
writings on malware, exploits, coding, & more...

- About
- #OBTS
- Book Series
- Objective-We
- Our Store/Swag
- Malware Collection

Never miss a post!
Subscribe to our **RSS feed**, or subscribe to our **newsletter**.

TCCing is Believing! Apple finally adds TCC events to Endpoint Security!
03/27/2025

Apple will bring TCC events to Endpoint Security in macOS 15.4. Here, we covers details, nuances, and provide PoC code for the new 'ES_EVENT_TYPE_NOTIFY_TCC_MODIFY' event.

continue reading »

Leaking Passwords (and more!) on macOS
03/20/2025

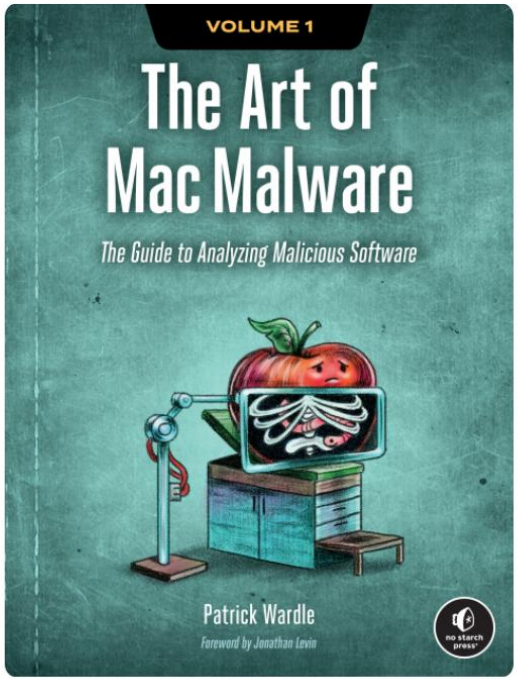
In this guest blog post, researcher Noah Gregory shares the technical details of a bug he uncovered (that was subsequently patched by Apple as CVE-2024-5447).

continue reading »

The Mac Malware of 2024
01/01/2025

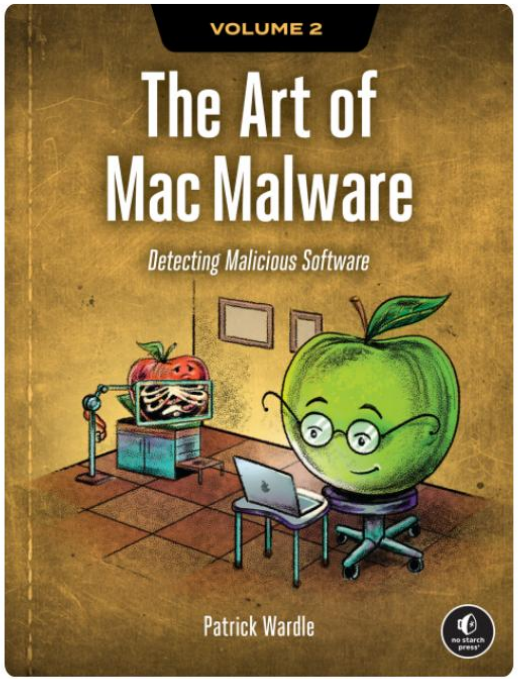
It's here! Our annual report on all the Mac malware of the year (2024 edition).

Besides providing samples for download, we cover infection vectors, persistence mechanisms, payloads, and more!



Volume I: Analysis

Defenders must understand how malware works to counter threats targeting Apple products. This volume explores



Volume II: Detection

Detecting malware is as vital as analyzing it. This volume explores programmatic approaches to detecting malicious