Return of the mac(OS): Transparency, Consent, and Control (TCC) Database Manipulation - Interpres Security

Archived: 2026-04-29 02:07:22 UTC

Summary

Discover insights into prominent malware campaigns, their connections to Democratic People's Republic of Korea (DPRK) adversary behaviors and gain valuable insights on threat hunting queries. This report will equip defenders with essential recommendations to safeguard against TCC.db abuse and stay ahead in the ever-changing cybersecurity landscape.

Preface

Historically, Windows has had a stronghold on the desktop market, and hence, security vendors focused their detection and prevention efforts primarily on Windows-based threats. However, over the years, corporations are increasingly leveraging MacBook laptops, and current trends indicate Apple is gaining market share, averaging 1% a year over the past 14 years (from 3% to 16.5%).

Source: Statcounter, accessed January 26, 2024

With macOS increasing in market presence, and in turn, eCrime and nation-state adversaries such as the Democratic People's Republic of Korea (DPRK)-attributed Lazarus pivoting to macOS, awareness of current and emerging macOS techniques is crucial to ensuring holistic prevention. This report will dive into some recently

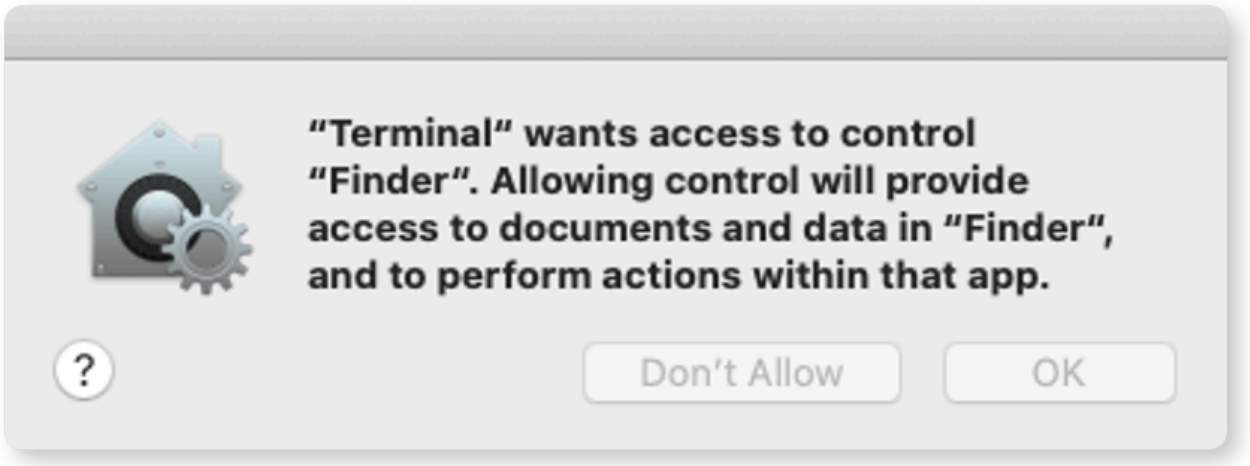
observed bypasses and tactics on macOS, how they are leveraged by adversaries and malware campaigns, with a behind-the-scenes look into my research and threat hunting, starting with the TCC database. Lastly, it will conclude with best practices and how defenders can detect or defend against these TTPs.

Primer on the TCC Database

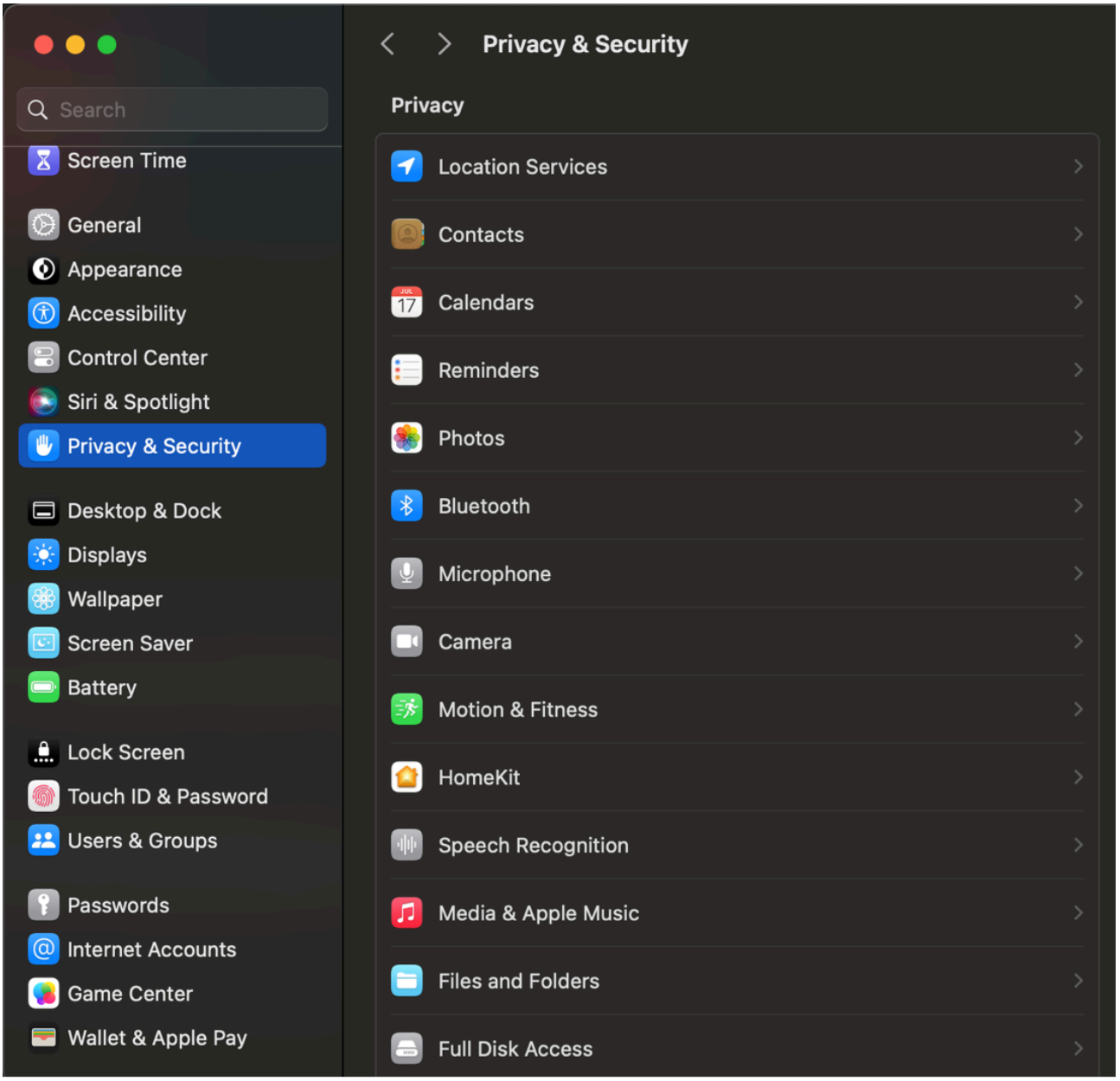
I referenced the TCC database in my MITRE ATT&CKCon; talk in 2023, so feel free to check that out.

For a deeper dive into the TCC framework, refer to Phil Stokes' writeup. For those of us new to macOS internals, here is an abridged version: The TCC framework is responsible for user security and privacy controls on Apple devices. There are two TCC databases on Apple devices, one at the system level: /Library/Application Support/com.apple.TCC/TCC.db, and one at the user level: $HOME/Library/Application Support/com.apple.TCC/TCC.db. The system level TCC database is protected by System Integrity Protection

(SIP), so it cannot be written to unless SIP is bypassed or disabled. The user level database can only be written to by a privileged process with proper entitlements, such as Full Disk Access (FDA).



User prompt to allow or deny an application access to a specific TCC protected item.



macOS Privacy & Security Objects

If you have a keen eye, you'll notice that TCC does not protect *all* folders and data on macOS -this will be important in later discussions.

Tccd is the dedicated for TCC; it exists for each logged-in user and system. This daemon is idle until it receives a request for an app to access TCC-protected data. The workflow is as follows:

macOS Tccd Workflow

Historical Bypasses and Limitations of TCC

Writing to TCC.db

The TCC framework on the surface seems like a win for security practitioners, but historically there have been many issues. I won't dive into all the bypasses, but if you are interested, I will list references at the bottom of this blog post for additional reading.

In earlier implementations, a malicious actor could directly write to TCC.db to give themselves permissions without prompting the user. If an app could write to TCC.db using sqlite3 INSERT commands, it can give itself all the relevant TCC entitlements. Thankfully, Apple addressed this issue with the implementation of System Integrity Protection (SIP) in macOS Sierra+. SIP was introduced as a security feature to prevent malicious files from modifying protected files.

In CVE-2020–9934, an actor could supply their own TCC.db. The gist of the bypass is as follows: an app cannot modify the database unless it has the entitlements: Com.apple.private.tcc.manager and com.apple.rootless.storage.TCC. If an actor finds a program with correct entitlements, they could control the TCC.db and use the tccd daemon to open an attacker-supplied TCC.db. Apple thankfully patched this Jul 15, 2020, in Security-Update 2020-004.

Similarly, in CVE-2020-27937, an actor can gain control of TCC by modifying the NFSHomeDirectory entry under the Directory Utility app. For context, the TCC daemon retrieves user information via opendirectoryd, and the corresponding directory utility, located at /System/Library/CoreServices/Applications/Directory Utility.app has the TCC entitlement (kTCCServiceSystemPolicySysAdminFiles) to allow it to modify the opendirectory files in /var/db/dslocal/nodes.

Therefore, a malicious actor could do the following to bypass the user TCC:

1

Copy the Directory Utility to a directory not protected by SIP.

2

Inject a malicious plugin (Mach-O bundle with a .daplug extension) to be executed with the Directory Utility's private TCC entitlements.

3

Prepare a fake TCC SQLite3 database with desired permissions.

4

Modify the NFSHomeDirectory to point to the fake TCC database.

5

Restart TCCd, and it will load the fake database based on the NFSHomeDirectory.

6

Profit!

What does TCC Protect?

As alluded to earlier, the TCC.db protects accessing specific folders, such as the Desktop and Documents. However, that does not prevent a user or unprivileged process from writing files to TCC "protected" areas and does not stop those files from being read. Therefore, malware authors could use these TCC protected areas as staging directories for their malware or secondary payloads.

```
user@macOShost ~ % ls Documents ls: Documents: Operation not permitted user@macOShost~ % echo malicious code here> Documents/nothing_to_see_here user@macOShost ~ % ls Documents ls: Documents: Operation not permitted user@macOShost ~ % cat Documents/nothing_to_see_here malicious code here
```

Areas TCC does NOT protect:

We talked about areas TCC does protect, though incomplete in implementation. I would be remiss if I did not briefly mention some sensitive directories on macOS that are not protected by TCC, such as:

```
$HOME/.ssh, $HOME/.aws, and /tmp, just to name a few.
```

These directories contain sensitive keys or can be used for staging, exfiltration, etc.

Problems with Full Disk Access (FDA)

One of the items encompassed by TCC is full disk access. If you are an admin, and you can grant yourself FDA, you will inherently grant ALL users (even non-admins!) the ability to read all other users' data on disk, including your own.

Mounting Issues

In CVE-2020-9771 a disk can be mounted and read by non-admin users. Apple "patched" this issue in macOS Catalina 10.15.4 in 2019 and in macOS Mojave in 2018, but the fix is tied to restricting Full Disk Access (FDA) permissions (kTCCServiceSystemPolicyAllFiles) for Terminal. However, the challenge in this "fix" is that in many environments, Terminal may have FDA enabled for legitimate reasons, and many apps, including IT and security software agents, may leverage scripting (Terminal) and require full disk access (FDA) to properly function and protect the endpoint.

Additionally, an actor could mount over ~/Library/Application Support/com.apple.TCC with their fake TCC.db by preparing a new TCC.db file. Because TCC privacy is not enforced on mounted local snapshots – this allows an actor to mount and access any file inside. Additionally, initiating and listing APFS snapshots is available for all users, so anyone can mount local apfs snapshots if using "noowners" flag.

```
mount_apfs -o noowners -s com.apple.-TimeMachine.2023-12-29-1 23456.local /Systems/Volumes/Data /tmp/snap
```

One Finder to Rule Them All

There is also an existing reflexive or chained "privilege escalation" via Finder. Finder, unbeknownst to the user, has access to FDA by default. If you are a keen investigator, you may notice that Finder is also not listed in your Security & Privacy permissions – it is transparent to the user. Also, if you grant Terminal access to Finder once, it is persistently granted. The user or admin would have to be aware of Finder's access and manually revoke permission. Therefore, an actor could procure Terminal control the Finder to secure FDA.

Social Engineering

Finally, social engineering also works!

Malware authors have been observed directing users on how to override Gatekeeper step-by-step, bypassing any additional security controls that admins deployed via MDM.

Example Step by Step Social Engineering

Enter the Lazarus Group

During my ATT&CKCon talk, I discussed this DPRK adversary in depth, as I tracked this nation-state's activities for a bit. The Lazarus Group is believed to operate out of North Korea, and has been active since 2009, gaining infamy over the years for high profile attacks against Sony, WannaCry, and recently JumpCloud, to name a few. With the poor state of the North Korean economy, Lazarus Group also is notable in their dual purpose of cyber espionage and nation-state financing. In 2023, they were heavily observed targeting cryptocurrency companies (no doubt, for said self-financing), and cross-platform attacks.

So why target macOS?

In recent years, Lazarus Group pivoted their targeting towards a more technical audience, having targeted security researchers in 2021 and engineers via social engineering methods using LinkedIn and email. With Operation Dream Job dating back to at least 2020, the Lazarus Group pretended to be cryptocurrency companies (as part of their cryptocurrency financing motives), targeting unsuspecting developers and engineering roles with the promise of lucrative (fake) jobs. In 2023, they continued this trend, impersonating large tech company recruiters, such as Amazon. With their targeting of engineers, I would expect their targets to be heavily leveraging macOS laptops.

Current desktop trends support this hypothesis. Stack Overflow conducted a survey in 2023, and out of over 87,000 developer responses, we see that macOS is gaining traction at 33%. With this shift, we are observing more adversaries adapting their tradecraft to be compatible with macOS (and Linux), and I would expect this trend to continue across adversaries.

Stack Overflow, 2023

Threat Hunting in VirusTotal

What do TCC interactions look like in the wild?

Important Disclaimer: I am by no means a VirusTotal query pro, and your mileage may vary depending on your subscription, date of query, etc.

Pulling on a Thread: Ties to North Korea

In tracking and researching Lazarus Group, I noted the heavy use of masquerading and TCC.db related activity. Specifically, I observed the following command line:

```
sqlite3 TCC.db SELECT kTCCServiceSystemPolicyFiles '.dump access'
```

Lazarus Group appears to be attempting to dump the access table from the TCC.db, likely to assess what applications had access to what permissions (FDA). From this command, I tried various iterations keying off of the dumped access table:

```
behaviour_processes:"TCC.db" AND (type:dmg or type:macho) AND behaviour:".dump"
```

Results: An inordinate number of hits 😮

Adding the access table as a criterion:

```
behaviour:access AND (type:dmg or type:macho) AND behaviour:".dump"
```

Results: 186 hits!

To my surprise, I received many hits for both queries, and wondered if this were a prolific but undocumented technique. However, upon examining this further, I realized VirusTotal was not using the quotes to denote a literal string, but instead treated the `.` as a wildcard, giving me thousands of unwanted results of any macOS file containing the "dump" string. Not to be defeated, I tried another angle. Knowing that the TCC.db is a SQLite3 database, that Lazarus leveraged bash and sqlite3 to run commands, and that many macOS malware will be DMG or Mach-O binaries, I searched for:

```
behaviour_processes:"bash -c sqlite3" (type:dmg OR type:macho) AND behaviour:tcc.db"
```

Results: 9 hits, all of which were popping up as "CloudMensis" malware. A quick google of CloudMensis gave me a Eureka moment.

CloudMensis is a macOS malware attributed to North Korean adversary APT37 (aka InkSquid, RedEyes, BadRAT, Reaper or ScarCruft). Pouring into the 9 samples I found on VT, CloudMensis consistently leverages csrutil to query the status of SIP protection, grepping for anything disabled, and proceeds to either directly INSERT into screencapture and FaceTime access or in other samples, add all relevant permissions line by line into the TCC.db. In parallel, the actor creates a new database in ~/Library/Application Support/com.apple.spotlight/Library/Application Support/com.apple.TCC/ and sets the HOME environment variable to ~/Library/Application Support/com.apple.spotlight using launchctl setenv. Lastly, it restarts tccd so that the TCC daemon loads the actor-controlled database.

```
/bin/bash /usr/bin/csrutil csrutil status /usr/bin/grep grep disabled /usr/bin/sqlite3 sqlite3 /Library/Application Support/com.apple. TCC/TCC.db INSERT INTO access VALUES('kTCCServiceScreenCapture', '/Users/user1/Library/Containers/com.apple.FaceTime/Data/Library/windowserver', 1, 1, 1, X'', NULL, NULL, 'UNUSED', NULL, NULL, 1711022347) launchctl setenv HOME %@ launchctl stop com.apple.tccd && launchctl start com.apple.tccd
```

CloudMensis Commands from samples in VirusTotal

Looking for command line parameters in VirusTotal, I leveraged this query:

```
content:com.apple.spotlight AND content:launchctl AND content:setenv AND content:com.apple.tccd AND (content:start and content:stop) NOT (tag:signed OR tag:legit)
```

Results: 35 hits total. Rummaging through the results, 8 hits tied back to CloudMensis.

CloudMensis uses two techniques to bypass TCC, and if successful, gains access to the screen, scans removable storage for documents of interest, and logs keyboard events. Performing screen capture requires bypassing TCC restrictions via exploiting CVE-2020-9934. Either the target hosts were known to be running macOS Catalina 10.5.6 or earlier, or the malware contained legacy code that the developers forgot to remove.

If SIP is disabled, CloudMensis adds entries to the TCC.db to grant itself permissions. If SIP is enabled but the Mac is running any version of macOS Catalina earlier than 10.15.6, CloudMensis will exploit a vulnerability to make the TCC daemon (tccd) load a database CloudMensis can write to (CVE-2020–9934)

Thankfully for defenders, CloudMensis malware seems to be mitigated if enterprises are using up-to-date MacBook's and SIP is enabled.

CloudMensis sample that uses INSERT statements line by line in the TCC.db

Casting A Wide Net: Keyloggers, Adware, Trojans, Oh My!

It is likely (if not inevitable) that attackers within the same nation-state will share code or adversarial practices. The DPRK is clearly no stranger to TCC abuse. I was curious how often other malware authors abused the TCC framework. I decided to start broadly in my queries, looking for ANY TCC interactions.

```
behaviour_processes:TCC.db
```

Results: 37 hits! Less than half were confirmed malware: Bundlore, CloudMensis again, Callisto (which I will detail below), the BlueBlood keylogger, and unspecified macOS Trojans the antiviruses comprising VirusTotal have yet to label.

I am assuming the BlueBlood keylogger samples I have corresponds to the BlueBlood keylogger that has been around for years. It uses INSERT or REPLACE to add access for appID, "com.applle.blblu," named in this fashion likely to masquerade as an Apple sanctioned appID for the accessibility list on macOS (kTCCServiceAccessibility).

Bundlore is a notorious macOS adware that has also been around since 2015 and gained notoriety in 2019. Bundlore, in older versions, will modify the TCC in a similar way to BlueBlood; i.e. if macOS version is 10.12 or older and Safari version is 10.0 or older, it modifies the TCC.db to enable AppleScript accessibility access to applications like Terminal, Safari, Apple Remote Desktop Agent, or Bash, so that it can interact with them.

In other Bundlore samples, I noted the use of the cp command to copy the entire TCC.db as a backup.

```
cp -f "/Library/Application Support/com.apple.TCC/TCC.db" "/Library/Application Support/com.apple.TCC/TCC.db.bak
```

Both Bundlore and BlueBlood leveraged insert and replace logic, so I narrowed my search to home in on these tactics, starting with REPLACE.

```
(type:dmg OR type:macho) and (behaviour:REPLACE and behaviour:sqlite3 and behavior:tcc.db) and NOT (tag:signed)
```

Results: 4 hits attributed to Callisto, a different macOS backdoor that some sources state serves as the forerunner of the Proton family of malware. Notably, like Bundlore, it leveraged insert and replace commands to manipulate the TCC.db.

Note: I reran the same query without the unsigned qualifier, and still got the 4 Callisto hits, which indicates that at the very minimum, the 4 Callisto samples were all unsigned. I'll dive into unsigned and ad hoc signed binaries in a future writeup.

It is worth noting that CoreServices folder (hidden in the System Library folder), which among other things, controls which apps have Accessibility authorization, became protected under SIP in 10.12 Sierra, so Callisto malware, like many malware that abuse TCC, will fail on SIP-protected machines running Sierra or later.

Repeating this query with INSERT commands:

```
(type:dmg or type:macho) and (behaviour:INSERT and behaviour:sqlite3 and behaviour:tcc.db) AND NOT (tag:signed)
```

Results: 11 hits, 10 of which were confirmed malware samples (but the "unconfirmed" sample looked highly suspicious).

Note: I added signature status again to test my hypothesis, and all samples were in fact unsigned.

Combining all of the observed TCC.db abuse commands:

```
(behaviour:tcc.db and behaviour:sqlite3) AND (behavior:SELECT OR behaviour:INSERT OR behaviour:REPLACE or behaviour:cp)
```

Results: 41 hits, less than half were confirmed malware. I'll take it!

Not on the Guest List

As I was pouring over samples and hits from my VT hunts, it dawned on me that a few malware campaigns that I know interacted with TCC were notably absent from my results.

JokerSpy is a malware leveraging SwiftBelt that targeted a cryptocurrency exchange in Japan, installing backdoors and deploying spyware. It creates a new TCC database and replaces the existing legitimate TCC database, leveraging the binaries xcc and cp.

Creation of actor-controlled TCC.db.
Source: Elastic

Interestingly enough, JokerSpy samples did not show up in my TCC.db query results. It is therefore likely that the copying of the TCC and related interactions via command line were decoupled from the JokerSpy malware itself, thus not triggering on VT.

XCSSET

While I was researching TCC bypasses, I came across XCSSET, another malware sample absent from my VT hunt results. Digging into how this campaign operates, the developers of XCSSET used three zero-days to bypass privacy protections, the last of which was used to bypass TCC framework, allowing them to take screen captures by hijacking the entitlements of other apps on the system. XCSSET malware is hardcoded with a list of apps it expects to have screen capture permissions, such as Zoom, Discord, WhatsApp, Teamviewer, etc. Using mdfind, it checks if the appIDs of those apps are present. If present, it will inject its own malicious app into the bundle of any of those apps, that way they can piggyback off the screen sharing permission granted to apps as part of their normal operation. From the XCSSET samples I found, the XCSSET AppleScript screenshot modules, downloaded from the malware author's command and control (C2) server were responsible primarily for the TCC bypass, and did not key off any TCC.db strings or sqlite3 commands.

If XCSSET fails to find or inject into those apps, it will impersonate Finder to ask user to grant Finder access. The malicious app can take screenshots or record the screen without needing explicit consent from the user because it inherits those TCC permissions from the parent app.

Recommendations for Blue Teamers:

- Obviously, use up-to-date systems and implement strict patching hygiene.
- If possible, disable FDA access to the terminal. If not possible, at minimum restrict access to the Terminal via an MDM solution or a security product.

- If possible, leverage an MDM solution to set privacy preferences via profiles.
- As a last (nuclear) option, you can reset TCC settings with tccutil

```
# Reset all permissions given to an application:
tccutil reset All app_id_here
# Reset the permissions granted to all apps:
tccutil reset All
```

- Enable Apple's protections, especially SIP. Configure security alerting to any disabling of SIP (typically accomplished via csrutil).
- Control permissions relating to software downloads or launches via MDM and/or application allow/deny lists via a security product.
- As always, user education is important. Educate your end users on common social engineering, and the repercussions of disabling security features like SIP.
- Exercise least privilege, only granting apps relevant permissions, and removing if no longer needed.
- Monitor sensitive directories that are not protected by TCC.

Conclusion

In this blog post, we have seen how the TCC has been abused and bypassed historically and continues to be a target by modern-day adversaries. In addition, we have looked at procedural examples of commands run by nation-state adversaries (in particular, the DPRK) and common macOS malware campaigns over the years, including adware, spyware, keyloggers, and trojans, and ways to detect these behaviors. We also examined how malware authors have leveraged interactive or decoupled commands to perform TCC.db manipulation, possibly as a means of file-based detection evasion.

Given the growth of macOS in the industry, it is critical for security teams to understand and secure enterprise across platforms.

Takeaways, Caveats, Final Thoughts

I went through a ton of threat hunting examples, and I would like to caveat that just because we did not see hits in VT does not mean these techniques are not happening in the wild. As we saw with my initial example from Lazarus Group dumping the TCC.db, it is likely that malware authors are decoupling their commands from their malware. VirusTotal will only show malware strings and commands, whereas adversaries can and will run interactive commands when they have access to the system. Yara signatures are also file-based in nature and will not detect fileless behaviors distinct and separate from malware. Having up-to-date antivirus definitions is not enough. Alert on command-line activities that may stem from interactive or decoupled activities.

Additionally, VirusTotal (and other sources) will change their signatures as more security vendors and community contributors flag malicious activity. These were my results at the time of queries run. Your mileage may vary depending on what type of VirusTotal account you have (free vs enterprise, paid version), and if you used free text search or "content," or "behaviour" search criteria.

Adversaries will pivot based on how the security and IT landscape changes, whether it be increasing macOS usage, migrating to the cloud, etc. It is important that security teams are not myopic in their defenses.

Appendix

Other Techniques – Discovery

Adversaries performed TCC.db privacy setting enumeration and discovery by using SELECT statements in sqlite3:

On macOS 11 BIG SUR and later:

```
sudo sqlite3 /Library/Application Support/com.apple.TCC/TCC.db “SELECT client,auth_value FROM access WHERE service==’kTCCServiceSystemPolicyAllFiles” | grep ‘2’$
```

On macOS 10.15 Catalina and earlier:

```
sudo sqlite3 /Library/Application Support/com.apple.TCC/TCC.db “SELECT client,auth_value FROM access WHERE service==’kTCCServiceSystemPolicyAllFiles” | grep ‘1’$
```

Coming Back Empty Handed from the (VirusTotal) Hunt

Querying for Full Disk Access

```
content:kTCCServiceSystemPolicyAllFiles or behaviour:kTCCServiceSystemPolicyAllFiles
```

3 hits: (at the time of discovery) a day-old Pearcleaner macOS trash cleaner opensourced on Github. It is unclear if this is a viable or high-fidelity query currently.

To mount or not to mount

I investigated the apfs mounting bypass I detailed earlier, noting the “noowners” flag.

```
(type:dmg or type:macho) behaviour:mount behaviour:noowners
```

Results: 6 hits, all benign.

```
behaviour:attach AND behavior:mountpoint AND behavior:com.apple.TCC
```

Results: 1hit, an Adobe update script

```
(type:dmg or type:macho) AND behaviour:mount AND behavior:apfs AND behaviour:noowners
```

Results: 4 hits, all benign.

Keying off of “noowners” and various mounting parameters, there did not seem to be a use case for this occurring in the wild. Notably, there were very few hits, and all were false positives. It is also possible that a malicious actor could do this interactively (or hands-on-keyboard), again decoupling it from the malware/file itself. However, given the low frequency in which this type of mounting behavior occurs, it still might be a worthwhile detection opportunity in an EDR or SIEM solution.
