Bypassing macOS TCC User Privacy Protections By Accident and Design - SentinelLabs

By Phil Stokes

Published: 2021-07-01 · Archived: 2026-04-05 19:11:26 UTC

Executive Summary

- TCC is meant to protect user data from unauthorized access, but weaknesses in its design mean that protections are easily overridden inadvertently.
- Automation, by design, allows Full Disk Access to be 'backdoored' while also lowering the authorization barrier.
- Multiple partial and full TCC bypasses are known, with several actively exploited in the wild.
- TCC does not prevent processes reading and writing to 'protected' locations, a loophole that can be used to hide malware.

Introduction

In recent years, protecting sensitive user data on-device has become of increasing importance, particularly now that our phones, tablets and computers are used for creating, storing and transmitting the most sensitive data about us: from selfies and family videos to passwords, banking details, health and medical data and pretty much everything else.

With macOS, Apple took a strong position on protecting user data early on, implementing controls as far back as 2012 in OSX Mountain Lion under a framework known as 'Transparency, Consent and Control', or TCC for short. With each iteration of macOS since then, the scope of what falls under TCC has increased to the point now that users can barely access their own data – or data-creating devices like the camera and microphone – without jumping through various hoops of giving 'consent' or 'control' to the relevant applications through which such access is mediated.

There have been plenty of complaints about what this means with regards to usability, but we do not intend to revisit those here. Our concern in this paper is to highlight a number of ways in which TCC fails when users and IT admins might reasonably expect it to succeed.

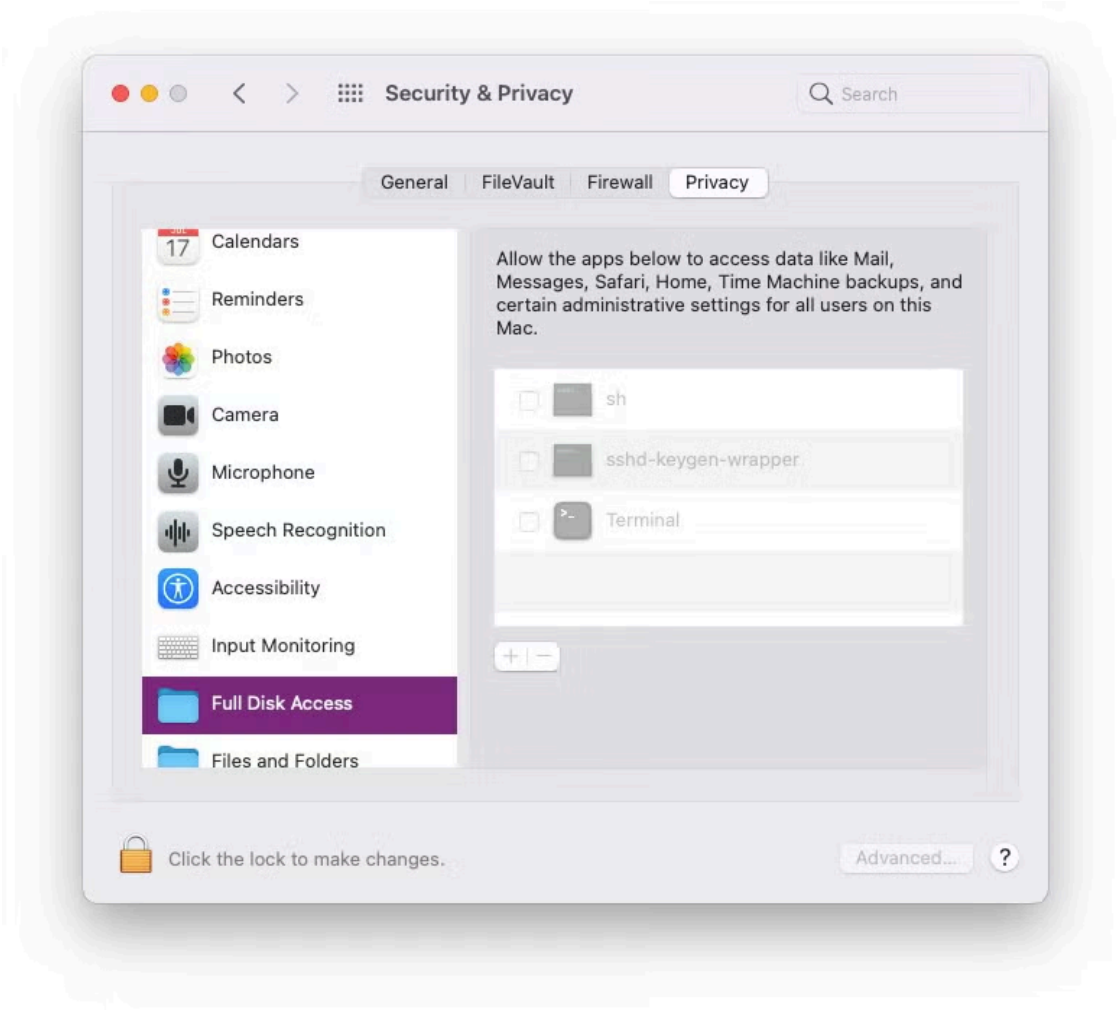
We hope that by bringing attention to these failures, users and admins might better understand how and when sensitive data can be exposed and take that into account in their working practices.

Crash Course: What's TCC Again?

Apple's latest platform security guide no longer mentions TCC by name, but instead refers to 'protecting app access to user data'. The current version of the platform security guide states:

“Apple devices help prevent apps from accessing a user’s personal information without permission using various technologies…[in] System Preferences in macOS, users can see which apps they have permitted to access certain information as well as grant or revoke any future access.”

In common parlance, we’re talking about privacy protections that are primarily managed by the user in System Preferences’ Privacy tab of the Security & Privacy pane.



System Preferences.app provides the front-end for TCC

Mac devices controlled by an MDM solution may also set various privacy preferences via means of a Profile. Where in effect, these preferences will not be visible to users in the Privacy pane above. However, they can be enumerated via the TCC database. The command for doing so changes slightly with Big Sur and later.

macOS 11 (Big Sur) and later:

```
sudo sqlite3 /Library/Application Support/com.apple.TCC/TCC.db "SELECT client,auth_value FROM access
```

macOS 10.15 (Catalina) and earlier:

```
sudo sqlite3 /Library/Application Support/com.apple.TCC/TCC.db "SELECT client,allowed FROM access WH
```

The command line also presents users and administrators with the `/usr/bin/tccutil` utility, although its claim to offer the ability "to manage the privacy database" is a little exaggerated since the only documented command is `reset`. The tool is useful if you need to blanket wipe TCC permissions for the system or a user, but little else.

```
tccutil(1)               BSD General Commands Manual              tccutil(1)

NAME
     tccutil -- manage the privacy database

SYNOPSIS
     tccutil command service [bundle_id]

DESCRIPTION
     The tccutil command manages the privacy database, which stores decisions the user has made about
     whether apps may access personal data.

     One command is currently supported:

     reset    Reset all decisions for the specified service, causing apps to prompt again the next time
              they access the service. If a bundle identifier is specified, the service will be reset for
              that bundle only.

EXAMPLES
     To reset all decisions about whether apps may access the address book:

           tccutil reset AddressBook
           tccutil reset All com.apple.Terminal

Darwin                          April 3, 2012                          Darwin
(END)
```

The spartan man page from tccutil

Under the hood, all these permissions are managed by the TCC.framework at `/System/Library/PrivateFrameworks/TCC.framework/Versions/A/Resources/tccd`.

```
MacBook-Pro :: Versions/A/Resources » strings tccd | grep -i kTCCService
kTCCServiceAll
kTCCServiceExposureNotification
kTCCServiceAccessibility
kTCCServiceFaceID
kTCCServiceSystemPolicyAllFiles
AlwaysAllowedService.kTCCServiceAppleEvents
AlwaysAllowedService.kTCCServiceAppleEvents preference
kTCCServiceAppleEvents
DELETE FROM access WHERE service = 'kTCCServiceAppleEvents' AND client LIKE 'com.apple.%'
DELETE FROM access WHERE client = 'com.apple.QuickTimePlayerX' AND service IN ('kTCCServiceCamera', 'kTCCServiceMicrop
hone', 'kTCCServiceScreenCapture')
DELETE FROM access WHERE client = 'com.apple.Health' AND service = 'kTCCServiceLiverpool'
DELETE FROM access WHERE client = '/usr/sbin/sshd' AND service = 'kTCCServiceSystemPolicyAllFiles'
kTCCServiceAddressBook
kTCCServiceContactsLimited
kTCCServiceContactsFull
kTCCServiceCalendar
kTCCServiceReminders
kTCCServiceTwitter
kTCCServiceFacebook
kTCCServiceSinaWeibo
kTCCServiceTencentWeibo
kTCCServiceShareKit
kTCCServiceLiverpool
kTCCServiceUbiquity
kTCCServicePhotos
kTCCServicePhotosAdd
kTCCServiceCamera
kTCCServiceMicrophone
kTCCServiceWillow
kTCCServiceMediaLibrary
kTCCServiceSiri
kTCCServiceMotion
kTCCServiceSpeechRecognition
kTCCServiceUserTracking
```

Strings in tccd binary reveal some of the services afforded TCC protection

Looked at in a rather narrow way with regard to how users work with their Macs in practice, one could argue that the privacy controls Apple has designed with this framework work as intended *when users (and apps) behave as intended* in that narrow sense. However, as we shall now see, problems arise when one or both go off script.

Full Disk Access – One Rule That Breaks Them All

To understand the problems in Apple's implementation of TCC, it's important to understand that TCC privileges exist at two levels: the user level and the system level. At the user level, individual users can allow certain permissions that are designed only to apply to their own account and not others. If Alice allows the Terminal access to her Desktop or Downloads folders, that's no skin off Bob's nose. When Bob logs in, Terminal won't be able to access Bob's Desktop or Downloads folders.

At least, that's how it's supposed to work, but if Alice is an admin user and gives Terminal Full Disk Access (FDA), then Alice can quite happily navigate to Bob's Desktop and Downloads folders (and everyone else's) regardless of what TCC settings Bob (or those other users) set. Note that Bob is not afforded any special protection if he is an admin user, too. Full Disk Access means what it says: it can be set by one user with admin rights and it grants access to all users' data system-wide.

While this may seem like good news for system administrators, there are implications that may not be readily apparent, and these implications affect the administrator's own data security.

When Alice grants FDA permission to the Terminal for herself, all users now have FDA permission via the Terminal as well. The upshot is that Alice isn't only granting herself the privilege to access others' data, she's granting others the privilege to access *her data*, too.

Surprisingly, Alice's (no doubt) unintended permissiveness also extends to unprivileged users. As reported in CVE-2020-9771, allowing the Terminal to have Full Disk Access renders all data readable without any further security challenges: the entire disk can be mounted and read even by non-admin users. Exactly how this works is nicely laid out in this blog post here, but in short any user can create and mount a local snapshot of the system and read all other users' data.

```
┌─[phils MacBook-Pro] - [/tmp] - [Fri Jun 04, 14:21]
└─[$] <> mkdir snap
┌─[phils MacBook-Pro] - [/tmp] - [Fri Jun 04, 14:21]
└─[$] <> tmutil localsnapshot
NOTE: local snapshots are considered purgeable and may be removed at any time by deleted(8).
Created local snapshot with date: 2021-06-04-142210
┌─[phils MacBook-Pro] - [/tmp] - [Fri Jun 04, 14:22]
└─[$] <> tmutil listlocalsnapshots /
Snapshots for disk /:
com.apple.TimeMachine.2021-06-04-142210.local
live_D3E81FDD-BD40-4368-980B-8BAA0A84A9F3
┌─[phils MacBook-Pro] - [/tmp] - [Fri Jun 04, 14:22]
└─[$] <> mount_apfs -o noowners -s com.apple.TimeMachine.2021-06-04-142210.local /System/Volumes/Data /tmp/snap
mount_apfs: snapshot implicitly mounted readonly
┌─[phils MacBook-Pro] - [/tmp] - [Fri Jun 04, 14:22]
└─[$] <> cd /tmp/snap; ls -al
total 0
drwxr-xr-x    21 phils  staff    672 May 31 15:31 .
drwxrwxrwt     9 root   wheel    288 Jun  4 14:23 ..
d--x--x--x     9 phils  staff    288 May 31 15:31 .DocumentRevisions-V100
drwxr-xr-x     5 phils  staff    160 May 31 15:29 .PreviousSystemInformation
drwx------     4 phils  staff    128 May  6 06:11 .Spotlight-V100
drwxr-xr-t     2 phils  staff     64 May 31 15:30 .TemporaryItems
drwx------  1344 phils  staff  43008 Jun  4 14:00 .fseventsd
drwxrwxr-x    31 phils  staff    992 Jun  1 00:34 Applications
drwxr-xr-x     7 phils  staff    224 May  6 06:09 Firmware
drwxr-xr-x    71 phils  staff   2272 May 31 15:30 Library
drwxr-xr-x@    3 phils  staff     96 Jan  1  2020 System
drwxr-xr-x     6 phils  staff    192 Jan  1  2020 Users
drwxr-xr-x     3 phils  staff     96 May 31 15:31 Volumes
```

Even Standard users can read Admin's private data

The 'trick' to this lies in two command line utilities, both of which are available to all users: `/usr/bin/tmutil` and `/sbin/mount`. The first allows us to create a local snapshot of the entire system, and the second to mount that

snapshot as an `apfs` read-only file system. From there, we can navigate all users data as captured on the mounted snapshot.

It's important to understand that this is not a bug and will not be fixed (at least, *'works as intended'* appears to be Apple's position at the time of writing). The CVE mentioned above was the bug for being able to exploit this *without* Full Disk Access. Apple's fix was to make it only possible when Full Disk Access has been granted. The **tl;dr** for Mac admins?

When you grant yourself Full Disk Access, you grant all users (even unprivileged users) the ability to read all other users' data on the disk, including your own.

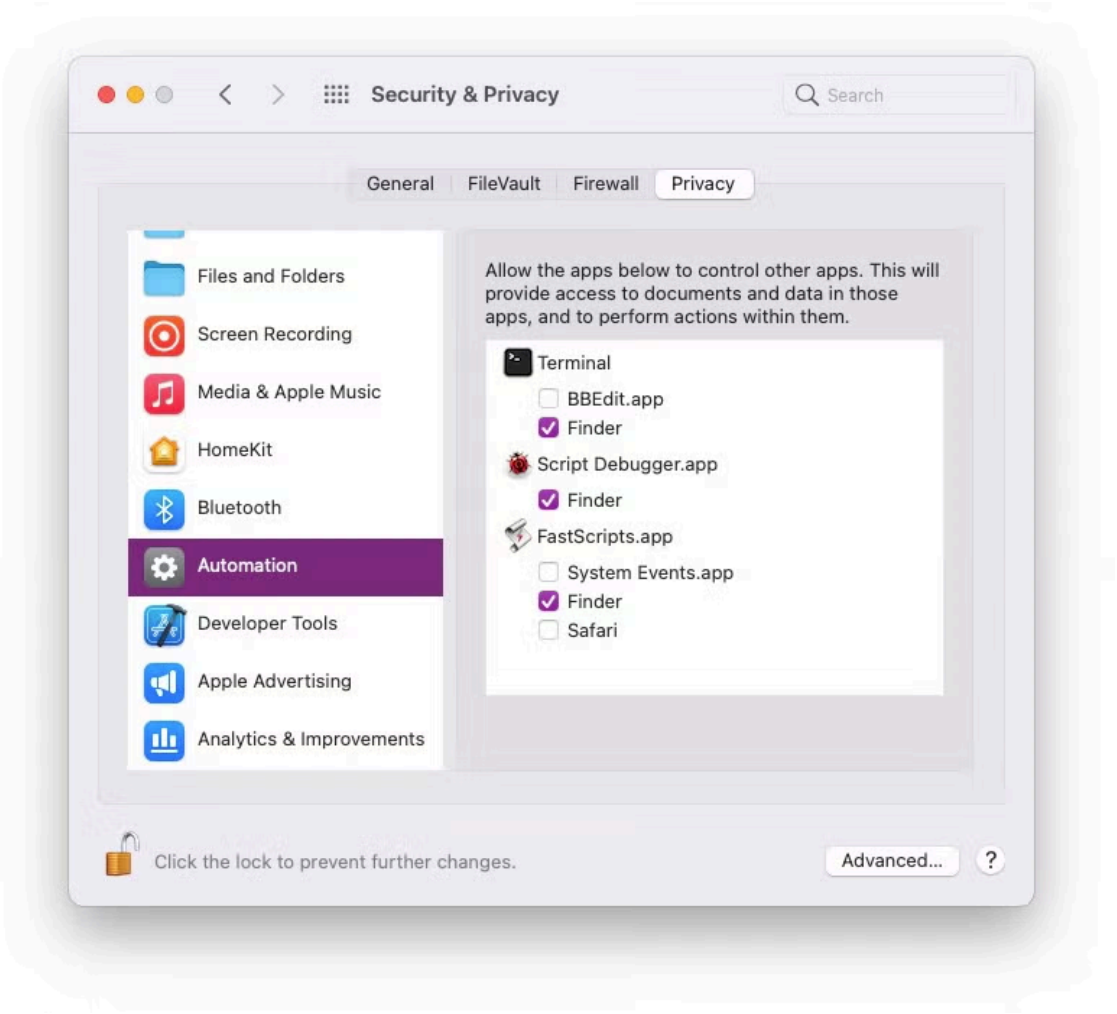
Backdooring Full Disk Access Through Automation

This situation isn't restricted only to users: it extends to user processes, too. Any *application* granted Full Disk Access has access to all user data, by design. If that application is malware, or can be controlled by malware, then so does the malware. But application *control* is managed by another TCC preference, Automation.

And here lies another trap: there is one app on the Mac that *always* has Full Disk Access but never appears in the Full Disk Access pane in System Preferences: the Finder.

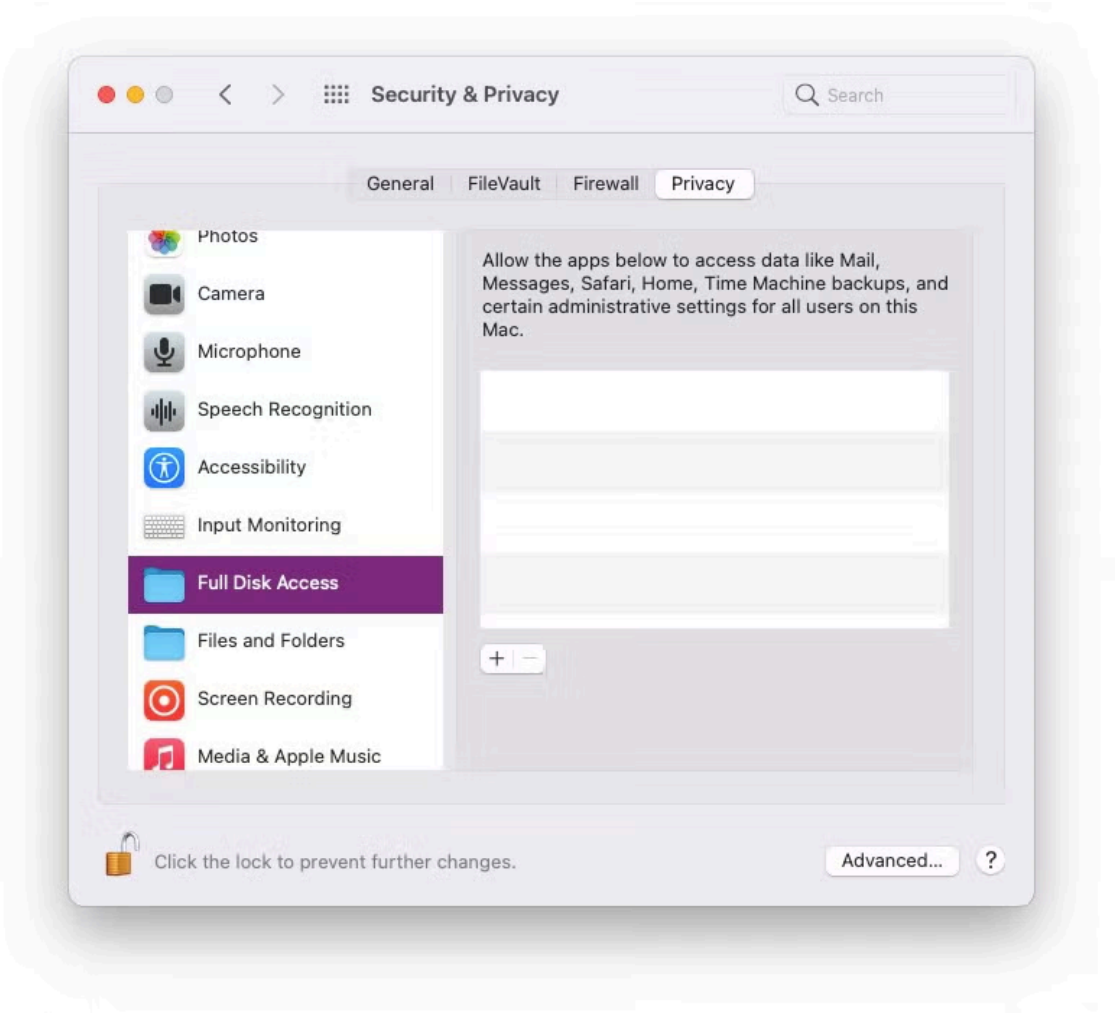
Any application that can control the Finder (listed in 'Automation' in the Privacy pane) also has Full Disk Access, although you will see neither the Finder nor the controlling app listed in the Full Disk Access pane.

Because of this complication, administrators must be aware that even if they never grant FDA permissions, or even if they lock down Full Disk Access (perhaps via MDM solution), simply allowing an application to control the Finder in the 'Automation' pane will bypass those restrictions.



Automating the Finder allows the controlling app Full Disk Access

In the image above, Terminal, and two legitimate third party automation apps, Script Debugger and FastScripts, all have Full Disk Access, although none are shown in the Full Disk Access privacy pane:



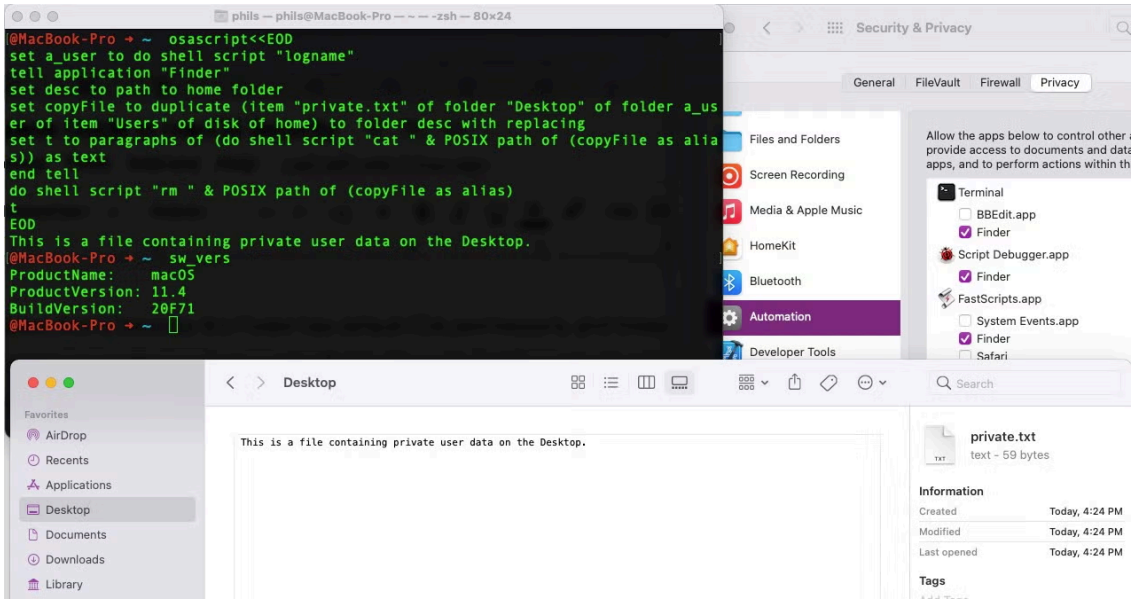
Apps that backdoor FDA through Automation are not shown in the FDA pane

As noted above, this is because the Finder has irrevocable FDA permissions, and these apps have been given automation control over the Finder. To see how this works, here's a little demonstration.

```
~  osascript<<EOD
set a_user to do shell script "logname"
tell application "Finder"
set desc to path to home folder
set copyFile to duplicate (item "private.txt" of folder "Desktop" of folder a_user of item "Users" o
set t to paragraphs of (do shell script "cat " & POSIX path of (copyFile as alias)) as text
end tell
do shell script "rm " & POSIX path of (copyFile as alias)
t
EOD
```

Although the Terminal is not granted Full Disk Access, if it has been granted Automation privileges for any reason in the past, executing the script above in the Terminal will return the contents of whatever the file "private.txt" contains. As "private.txt" is located on the user's Desktop, a location ostensibly protected by TCC, users might

reasonably expect that the contents of this file would remain private if no applications had been explicitly granted FDA permissions. This is demonstrably not the case.

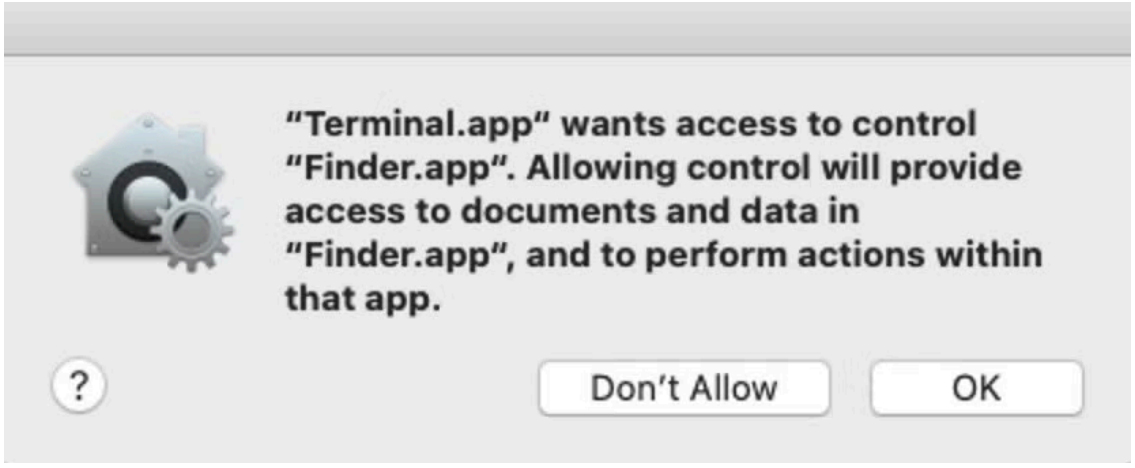


Backdooring FDA access through automating the Finder

The obvious mitigation here is not to allow apps the right to automate the Finder. However, let's note two important points about that suggestion.

First, there are many legitimate reasons for granting automation of the Finder to the Terminal or other productivity apps: any mildly proficient user who is interested in increasing their productivity through automation may well have done so or wish to do so. Unfortunately, this is an "All-In" deal. If the user has a specific purpose for doing this, there's no way to prevent other less legitimate uses of Terminal's (or other programs') use of this access.

Second, backdooring FDA access in this way results in a lowering of the authorization barrier. Granting FDA in the usual way requires an administrator password. However, one can grant consent for automation of the Finder (and thus backdoor FDA) without a password. A consent dialog with a simple click-through will suffice:



A simple 'OK' gives access to control the Finder, and by extension Full Disk Access.

While the warning text is explicit enough (if the user reads it), it is far from transparent that given the Finder's irrevocable Full Disk Access rights, the power being invested in the controlling app goes far beyond the current user's consent, or control.

As a bonus, this is not a per-time consent. If it has ever been granted at any point in the past, then that permission remains in force (and thus transparent, in the not-good sense, to the user) unless revoked in System Preferences 'Automation' pane or via the previously mentioned `tccutil reset` command.

The **tl;dr**: keep a close and regular eye on what is allowed to automate the Finder in your System Preferences Privacy pane.

The Sorry Tale of TCC Bypasses

Everything we've mentioned so far is actually by design, but there is a long history of TCC bypasses to bear in mind as well. When macOS Mojave first went on public release, SentinelOne was the first to note that TCC could be bypassed via SSH (this finding was later duplicated by others). The indications from multiple researchers are that there are plenty more bypasses out there.

The most recent TCC bypass came to light after it was discovered being exploited by XCSSET malware in August 2020. Although Apple patched this particular flaw some 9 months later in May 2021, it is still exploitable on systems that haven't been updated to macOS 11.4 or the latest security update to 10.15.7.

On a vulnerable system, it's trivially easy to reproduce.

1. Create a simple trojan application that needs TCC privileges. Here we'll create an app that needs access to the current user's Desktop to enumerate the files saved there.

   ```
   % osacompile -e 'do shell script "ls -al /Users/sphil/Desktop >> /tmp/lsout"' -o /tmp/ls.app
   ```

2. Copy this new "ls.app" trojan to inside the bundle of an app that's already been given TCC permission to access the Desktop.

   ```
   % cp -R /tmp/ls.app /Applications/Some Privileged.app/
   ```

   One way you can find the current permitted list of apps is from the 'Files and Folders' category in the Privacy tab of System Preferences' Security & Privacy pane (malware takes another route, as we'll explain shortly).

3. Execute the trojan app:

   ```
   % open /Applications/Some Privileged.app/ls.app
   ```

Security-minded readers will no doubt be wondering how an attacker achieves Step 2 without already having knowledge of TCC permissions – you can't enumerate the list of privileged apps in the TCC.db from the Terminal

unless Terminal already has Full Disk Access.

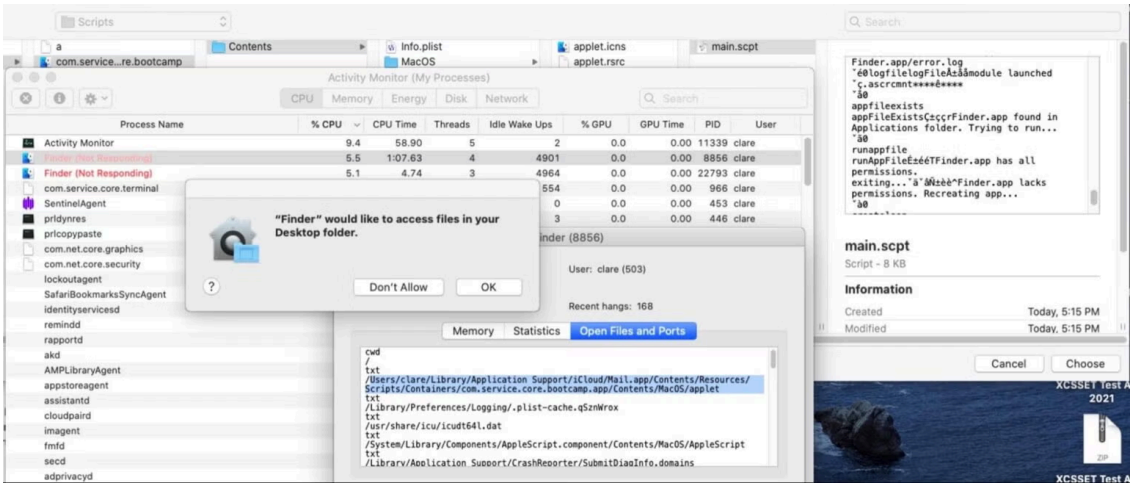
Assuming the target hasn't already granted Terminal FDA privileges for some other legitimate reason (and who hasn't these days?), an attacker, red teamer or malware could instead enumerate over the contents of the `/Applications` folder and take educated guesses based on what's found there, e.g., Xcode, Camtasia, and Zoom are all applications that, if installed, are likely to be privileged.

Similarly, one could hardcode a list of apps known to have such permissions and search the target machine for them. This is precisely how XCSSET malware works: the malware is hardcoded with a list of apps that it expects to have screen capture permissions and injects its own app into the bundle of any of those found.

```
00033 PushLiteral 9 #  ;  String: 'us.zoom.xos'

00034 PushLiteral 10 #  ;  String: 'com.hnc.Discord'

00035 PushLiteral 11 #  ;  String: 'WhatsApp'

00036 PushLiteral 12 #  ;  String: 'com.tinyspeck.slackmacgap'

00037 PushLiteral 13 #  ;  String: 'com.tencent.xinWeChat'

00038 PushLiteral 14 #  ;  String: 'com.teamviewer.TeamViewer'

00039 PushLiteral 15 #  ;  String: 'com.upwork.Upwork'

0003a PushLiteralExtended 16 #  ;  String: 'com.parallels.desktop.console'

0003d PushLiteralExtended 17 #  ;  String: 'com.parallels.desktop.appstore'

00040 PushLiteralExtended 18 #  ;  String: 'com.screenshotmonitor.SSM-App'

00043 PushLiteralExtended 19 #  ;  String: 'com.bohemiancoding.sketch3'

00046 PushLiteralExtended 20 #  ;  String: 'com.skype.skype'
00049 PushLiteralExtended 21 # <Value type=fixnum value=0xc>   ;  Decimal value = 12
```

Decoded strings from XCSSET malware reveals a list of apps it exploits for TCC permissions

Unfortunately, the fix for this particular bug doesn't effectively stop malware authors. If the bypass fails, it's a simple matter to just impersonate the Finder and ask the user for control. As with the Automation request, this only requires the user to click-through their consent rather than provide a password.



Fake Finder App used by XCSSET malware to access protected areas

As we noted above, the (real) Finder already has Full Disk Access by default, so users seeing a request dialog asking to grant the Finder access to any folder should immediately raise suspicion that something is amiss.

TCC – Just One More Thing

That almost wraps up our tour of TCC gotchas, but there's one more worth pointing out. A common misunderstanding with Apple's User privacy controls is that it prevents access to certain locations (e.g., Desktop, Documents, Downloads, iCloud folders). However, that is not quite the case.

Administrators need to be aware that TCC doesn't protect against files being written to TCC protected areas by unprivileged processes, and similarly nor does it stop files so written from being read by those processes.

```
➜  ~ cd ~/Desktop
➜  Desktop ls -al
ls: .: Operation not permitted
➜  Desktop echo 'I can still read and write files to the Desktop and other protected areas' > ~/Desktop/newfile
➜  Desktop ls -al
ls: .: Operation not permitted
➜  Desktop cat newfile
I can still read and write files to the Desktop and other protected areas
➜  Desktop sw_vers
ProductName:    macOS
ProductVersion: 11.4
BuildVersion:   20F71
➜  Desktop
```

A process can write to a TCC protected area, and read the files it writes

Why does this matter? It matters because if you have any kind of security or monitoring software installed that doesn't have access to TCC-protected areas, there's nothing to stop malware from hiding some or all of its components in these protected areas. TCC isn't going to stop malware using those locations – a blind spot that not every Mac sys administrator is aware of – so don't rely on TCC to provide some kind of built-in protected 'safe-zone'. That's not how it works, when it works at all.

Conclusion

We've seen how macOS users can easily and unknowingly expose data they think is protected by TCC simply by doing the things that macOS users, particularly admins, are often inclined to do. Ironically, most of these 'inadvertent breaches' are only possible because of TCC's own lack of transparency. Why, for example, is the Finder not listed in the Full Disk Access pane? Why is it not clear that Automation of the Finder backdoors Full Disk Access? And why is password-authentication downgraded to a simple consent prompt for what is, effectively, the same privilege?

Other questions raised by this post concern whether consent should have finer grained controls so that prompts can be optionally repeated at certain intervals, and – perhaps most importantly – whether users should be able to protect their own data by being allowed to opt out of FDA granted by other users on the same device.

We know that malware abuses some of these loopholes, and that various TCC bugs exist that have yet to be patched. Our only conclusion at this point has to be that neither users nor admins should place too much faith in the ability of TCC as it is currently implemented to protect data from unauthorized access.

Source: https://www.sentinelone.com/labs/bypassing-macos-tcc-user-privacy-protections-by-accident-and-design/