

Bypassing TCC on macOS

Rohit Chatterjee

1 Overview

I have found a vulnerability in the macOS System Migration framework that allows a program running as root without any existing TCC permissions to silently gain the Full Disk Access TCC permission. Since programs with Full Disk Access can modify any user's TCC database, this vulnerability also allows programs to silently gain any user-specific TCC permission, including camera and microphone access. I created a proof-of-concept (PoC) exploit that runs a specified command with Full Disk Access, along with a script that grants camera and microphone access to a specified app. Both have been tested on a clean install of macOS 12.4.

pre>admin@admins-MacBook-Pro tcc-bypass % sudo ls "/Library/Application Support/com.apple.TCC"
[Password:
ls: /Library/Application Support/com.apple.TCC: Operation not permitted
admin@admins-MacBook-Pro tcc-bypass % sudo ./get-full-disk-access.sh ls "/Library/Application Support/com.apple.TCC"
SSH daemon started. Waiting 10 seconds...
Migration request sent. Waiting 30 seconds...

----- Output: -----

AdhocSignatureCache
TCC.db

Done.
admin@admins-MacBook-Pro tcc-bypass %

Figure 1: The PoC lists the contents of the directory containing the system-wide TCC database, which is not readable without Full Disk Access.

```
tcc-bypass --zsh -- 130x25
admin@admins-MacBook-Pro tcc-bypass % sudo ./get-full-disk-access.sh ./grant-cam-mic-access.sh "/Applications/Google Chrome.app"
SSH daemon already loaded. Continuing.
SSH daemon already has Full Disk Access. Continuing.

----- Output: -----
Granted camera and microphone access to /Applications/Google Chrome.app

-----
Done.
admin@admins-MacBook-Pro tcc-bypass %
```

Figure 2: The PoC runs the included script to grant camera and microphone access to Chrome.

2 The Exploit

The PoC first grants Full Disk Access to `sshd-keygen-wrapper` (the SSH daemon on macOS) using `systemmigrationd`, a binary with the `com.apple.rootless.install.heritable` entitlement. This gives Full Disk Access to any SSH connection to the system, so the PoC then runs a payload with Full Disk Access through the system’s SSH client.

The System Migration framework on macOS lets users transfer their files, apps, and settings from another computer or Time Machine backup. This framework processes migration requests through `systemmigrationd`, which loads a plugin, `TCCmigration`. This plugin, as its name suggests, migrates TCC settings. After decompiling the plugin in Ghidra, I found something that caught my attention.

```
uVar6 = __auth_stubs::_SMJobIsEnabled(uVar8, "com.openssh.sshd", &local_91);
...
iVar2 = __auth_stubs::_TCCAccessSetForPath(
    *(undefined8 *)__got::_kTCCServiceSystemPolicyAllFiles,
    "/usr/libexec/sshd-keygen-wrapper", uVar6);
```

The second line grants Full Disk Access (`kTCCServiceSystemPolicyAllFiles`) to the SSH daemon, but only if it is already loaded, which is checked by the first line. The PoC loads the SSH daemon with the following shell command:

```
launchctl load -w /System/Library/LaunchDaemons/ssh.plist
```

A migration can be started programmatically by placing a file in the directory `/Library/SystemMigration/Queue`. Any program with root privileges can do this, since the directory is not protected by SIP. The file must contain a migration request encoded as a "property list," or **plist**, containing information about the request, including its type, what data to transfer, and the source system to transfer from. The `TCCMigration` plugin only runs if the request's "type" property has a value of either 1 or 4. If the source system is set to a random UUID, then the migration fails, but not before running `TCCMigration` and granting Full Disk Access to the SSH daemon.

Once the SSH daemon has Full Disk Access, a malicious program on the system can gain Full Disk Access by logging into any user via SSH. By default, the SSH daemon requests the user's password, which the program does not necessarily know. The PoC gets around this by generating a SSH key pair and saving the public key to `~/.ssh/authorized_keys`. To regain root privileges inside the SSH session, the PoC executes its payload through a copy of `/bin/zsh` that has the `setuid` bit set.

To summarize, the PoC first loads the SSH daemon, then creates a file containing a malicious migration request and places it in `/Library/SystemMigration/Queue`, which starts `systemmigrationd`. Once the SSH daemon has Full Disk Access, the PoC executes its payload in a SSH session.

This vulnerability can be fixed by protecting the directory `/Library/SystemMigration` under SIP and requiring an entitlement (like `com.apple.rootless.storage.SystemMigration`) to modify it.