Firstline Privacy Defense: an insight into Apple's Transparency, Consent, Control framework

Osama Alhour (`https://github.com/NSAntoine`)
Computer Science, German Jordanian University

October 18, 2024

Contents

1 Introduction

In an era where tension and anxiety about user privacy and security are higher than they've ever been, where attacks are getting more sophisticated and there's more on the line in terms of user data than there has ever been before, tech companies are spending billions upon billions of dollars to secure and protect critical user data. Perhaps the most notable of which is Apple, going so far to protect user data as creating entirely new hardware such as the Secure Enclave Processor (on the iPhone 5s and later models), a physical co-processor in charge of important security concerns such as biometrics. It is no coincidence that Apple has positioned itself as a a leader in protecting user data and privacy with how many mitigations and privacy centered features they maintain (and continuously add year after year), one of the most important of which is Apple's Transparency, Consent, Control framework (commonly known and referred to as "TCC") framework, which regulates what and how apps interact with sensitive user data & device hardware by requiring applications to seek user permission to access desired data or services, whether the data is stored on-device (Photos, Contacts) or access to device hardware features (Camera, Microphone).

TCC addresses the extremely important concern of keeping critical user data safe by letting users control not just which apps access certain sensitive information & hardware features, but also *how* applications access the data they're being granted (i.e., users can pick *which* specific photos an application can access, rather than granting access to *all* photos), preventing applications from accessing data which the user wants to keep private from said application.

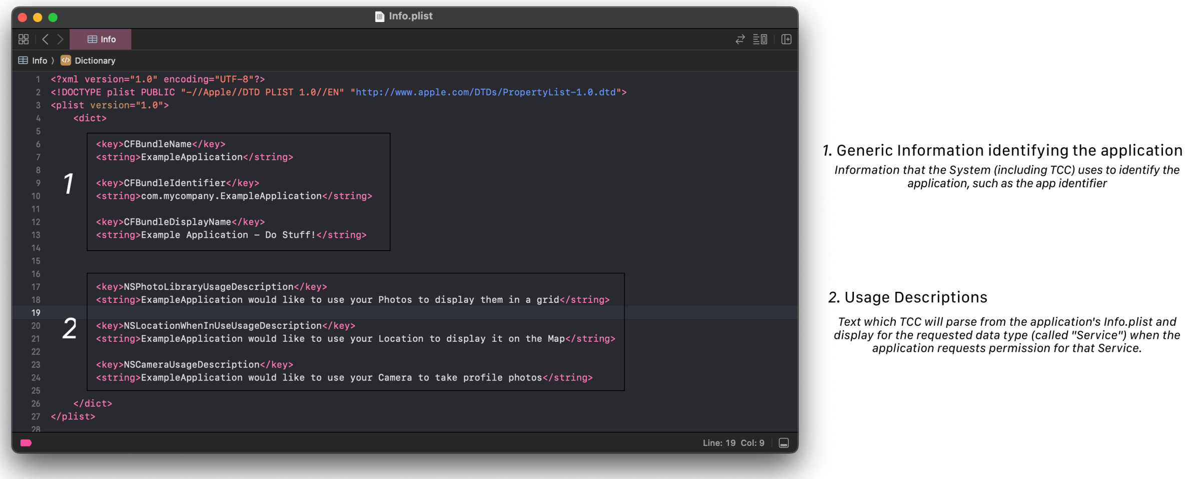
This paper seeks to provide a comprehensive technical analysis of how TCC works internally, how it interacts with other system components, and it's impact on both how developers shape their applications as well as user control granting sensitive data to applications.

2 User Interaction

One of, if not, *the most* important aspect of TCC is user control. The ability to control *if* an application has access to specific data, and, in the cases of some data, being able to control *what* data is accessible. To the user, it accomplishes this in a simple manner: a plain alert. In fact, when you get those alerts asking you "*Application X* would like to access your contacts/microphone/photos/etc...", that's TCC. To ensure this is secure, however, the alert isn't rendered by the actual application requesting those permissions, the application's role is just to request those permissions and get notified for if the user granted them or not. The alert is managed, presented, and drawn by separate processes on both iOS and macOS, ensuring that the process requesting the permissions cannot programmatically get ahold of the alert and simulate touch events or directly call functions that indicate the user pressed the "Allow" button.

In recent years, as Apple has been cracking down harder and harder on misuse of user data and privacy. TCC alerts have gotten a bit more sophisticated, where some now allow for the users to pick *specific* data that the application should be allowed to access. For example, starting in iOS 14, users can pick a specific subset of photos and videos to share with the applications requesting access to the Photos library[1].

When requesting access to a specific set of data / hardware features policed by TCC (Photos, Camera, Contacts, etc...), applications must specify and justify to the user *why* they want access to that data, applications store the text justifying why they want access to the data in a file called the `Info.plist`, which stores various other basic information about the application itself (i.e., bundle identifier, name of the application, application version, etc.). This justification text is later retrieved by the System and shown in the alert requesting permission. If an application tries to request data that it hasn't provided a description for (i.e., an application requests access to the user's Photos without providing a reason why in it's `Info.plist` file), the System terminates the application, and the application is guaranteed to be rejected from the App Store upon review. The justification text being in a static file also allows App Store review to ensure that a different text isn't shown at runtime when the alert is presented, since applications are not allowed to modify their `Info.plist` files.



```xml
<?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>CFBundleName</key>
        <string>ExampleApplication</string>

        <key>CFBundleIdentifier</key>
        <string>com.mycompany.ExampleApplication</string>

        <key>CFBundleDisplayName</key>
        <string>Example Application - Do Stuff!</string>



        <key>NSPhotoLibraryUsageDescription</key>
        <string>ExampleApplication would like to use your Photos to display them in a grid</string>

        <key>NSLocationWhenInUseUsageDescription</key>
        <string>ExampleApplication would like to use your Location to display it on the Map</string>

        <key>NSCameraUsageDescription</key>
        <string>ExampleApplication would like to use your Camera to take profile photos</string>

    </dict>
</plist>
```

1. Generic Information identifying the application

Information that the System (including TCC) uses to identify the application, such as the app identifier

2. Usage Descriptions

Text which TCC will parse from the application's Info.plist and display for the requested data type (called "Service") when the application requests permission for that Service.

Figure 1: Example of an application's Info.plist which includes descriptions for requesting user sensitive data, later parsed and displayed by TCC

3 TCC Who?

When an application on iOS / macOS needs access to user sensitive data or hardware features guarded by TCC, it is always provided by a System framework (a library). For example, Photos from the user's gallery are provided by the PhotosKit framework, access to the Camera is provided by the AVFoundation framework, and access to user contacts are provided by the Contacts framework. This means that the developer *never* has to interface with TCC themselves or know about it, in fact, the vast majority of iOS & macOS developers don't know about TCC, or what it does, or how it works, etc. This is information that is intentionally abstracted away from both the developer and the user. Apple writes frameworks which interact with TCC in order to allow developers to check and request authorization to data guarded by TCC (as TCC is considered *private API*, subject to be internally changed at any time, meaning it is not stable to have it's interface publically exposed to app developers).

4 Services

Data / Hardware features protected by TCC are called TCC ”Services”, they’re identified as string constants prefixed with `kTCCService...`, below is a table of commonly used & accessed TCC Services (Note that this doesn’t include *all* TCC Services, just commonly used ones):

Common TCC Services	
Service name	Data / Hardware features gated by TCC
kTCCServicePhotos	Access to user’s Photo Library
kTCCServiceCamera	Access to user’s Camera
kTCCServiceMicrophone	Access to user’s Microphone
kTCCServiceAddressBook	Access to user’s Contacts
kTCCServiceLiverpool	Access to user’s Location (used on iOS only)
kTCCServiceFaceID	Access to FaceID related capabilities
kTCCServiceScreenCapture	Access to Screen Capture capabilities
kTCCServicePostEvent	Access to capabilities related to synthetically posting HID events (i.e, synthetic key / touch presses)

Apple’s use of TCC is so extensive that there are exactly *100* TCC Services defined as of macOS 15.0.

5 Architecture

TCC is made up of 3 important parts within the System:
- `TCC.framework`: A private System framework which allows for interaction with TCC, including request permissions and checking for an application has been granted authorization to TCC Services
- `tccd`: Star of the show, daemon which manages permissions, authorization requests, the TCC database, and all actions to do with TCC.
- The TCC database(s)*: To be discussed further.

The TCC framework is used by other System frameworks (mostly public ones, such as PhotoKit and Contacts) to check authorization status as well as request authorization from the user, for example, the Contacts framework provided to user applications uses this framework in order to provide applications with information pertaining to if the user has granted Contacts permissions, as well as methods to request permissions from the user to access their Contacts, this pattern is observed with almost every single framework which provides access to TCC Services to applications.

The framework doesn't actually do any of the heavy lifting in relation to TCC, rather, it acts as a frontend to allow applications to interact with the TCC daemon, `tccd`, which *does* perform all the heavy lifting. The next section covers `tccd`, what it does, how it works, and it's importance.

6 `tccd` & The databases

The star of the show in terms of TCC is `tccd` (the TCC daemon), `tccd` acts as a server which receives requests by applications to perform actions related to TCC, such as requesting authorization when an application requests it, and carries those actions out. on every System, there's 2 `tccd` processes are running:

- A *"system"* `tccd` instance (macOS only!).
- A `tccd` instance for the current desktop user running.

on iOS, watchOS, and tvOS, the notion of a "desktop user" doesn't exist, on those platforms there's just the "mobile" user, therefore, on those platforms there is just one `tccd` process running. This is in contrast to macOS, where there's a *"system"* `tccd` process which runs as root as well as a `tccd` process for the current desktop user. The reason for the existence of a separate *"system"* `tccd` instance on macOS is to be discussed later in this section.

The role of `tccd` is quite simple, it acts as the security middleman between applications which request access to services and between the system which provides those services, the daemon provides services for applications to check if they have permissions for a certain service and to request authorization for that service

Clients communicate with `tccd` over Apple's primary framework for IPC called XPC, where `tccd` listens for connection requests by clients made to `tccd`'s service name `com.apple.tccd` (for the user mode `tccd` instance) and `com.apple.tccd.system` for the system instance.

Once clients have established a connection with the daemon, they send what is called a "message" to `tccd` containing a dictionary (hashmap) of keys and values for the daemon that it uses as context to perform actions for. All messages sent to `tccd` *must* include a key named `function` specifying what the client wants `tccd` to do. For example, when an application needs to request permissions for a specific TCC Service, it specifies the function key as *TCCAccessRequest*, and populates other keys, such as the key specifying which service it needs to request access to.

So how does `tccd` itself work anyway? How does it check for if an application already has permission for a specific service so applications don't need to prompt the user for permission to services every single time the application is used? How does it prompt the user in a secure manner? The answer to the first question is the TCC Database(s): iOS/watchOS/tvOS devices have just one database, while macOS has multiple: each desktop user has their own TCC database in order to ensure that permissions aren't shared between different users for the same applications, as well as TCC database used for the system `tccd`, which is the same no matter the user. On macOS, the user TCC database is always used *except* for the following services:

Service permissions stored in the System-wide TCC Database	
Service name	Data / Hardware features gated by TCC
kTCCServiceDeveloperTool	Access to Xcode Developer Tools
kTCCServiceSystemPolicyAllFiles	Full Disk Access
kTCCServiceAccessibility	Access to Accessibility features
kTCCServiceScreenCapture	Access to Screen Capture capabilities
kTCCServiceListenEvent	Access to listen to HID events such as touches and keystrokes
kTCCServicePostEvent	Access to capabilities related to synthetically posting HID events (i.e, synthetic keypresses)

Since these are recorded in the System-wide TCC database on macOS, it means the permissions for these services are shared across desktop accounts on the same computer, for example, if one user (say, user A) granted Screen Capture capabilities to Application foo, then Application foo would have Screen Capture capabilities on another desktop user on the same computer (say, user B).

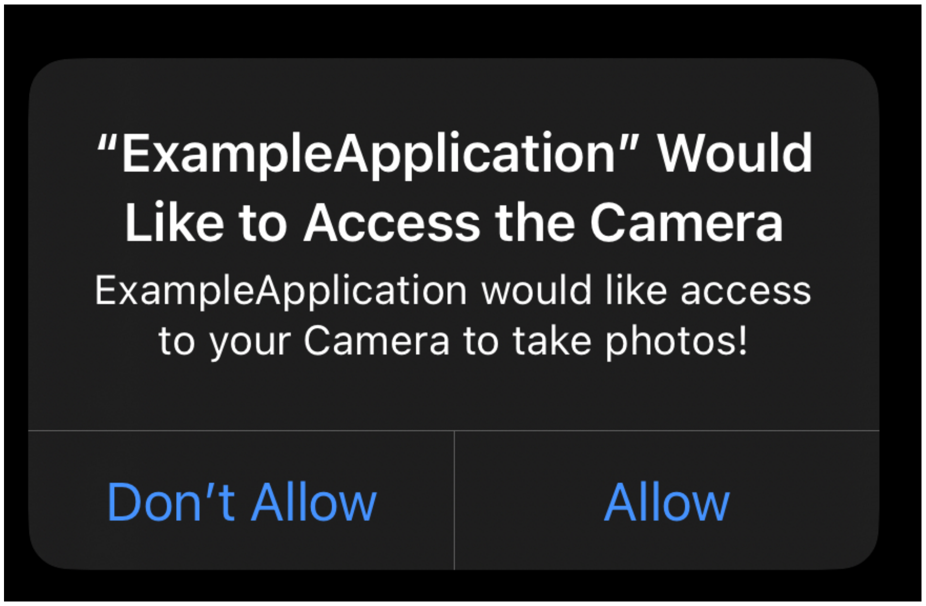
TCC Databases are SQLite3 databases which have multiple tables, but just one is of interest to us: the `access` table, the `access` table holds records of applications which have been given or denied access to TCC Services as well as which services those applications have been granted access to, and each row contains the following set of keys & data (note that this isn't a list of all keys, but rather just the most important):

`access` Table Fields		
Key	Type	Purpose
service	Text	The service which this application has been granted or denied
client	Text	String identifying the application (could be either bundle ID or absolute path[3])
client_type	Number	A number identifying the type of the `client` field (0 for if it's a bundle identifier, 1 for if it's the absolute path)[3].
auth_value	Number	A number identifying whether the service has been approved for this application or not. Possible values are: denied (0), unknown (1), allowed (2), or limited (3). [3]
auth_reason	Number	A number identifying the reason why auth_value is set to it's current value (i.e., a auth_reason of 3 indicates that `auth_value` is set to it's current value because of the user's choice). The full list of possible auth_reason values can be found on Keith Johnson's A deep dive into macOS TCC.db blog post
cs_req	BLOB (raw data)	Data which is translated into a *Code Signing Requirement Blob*, a String which describes criteria that the application needs to meet in terms of Code Signing (i.e, who is this application signed by?) to be granted access to this service. This is in order to avoid attacks where an application that doesn't have access to a particular service uses the same bundle ID as an application which does have access to the service (as those 2 applications with the same bundle identifier would have a different Code Signature)

But why is `tccd` even a thing? Well, normal applications do not have permission to read, touch, or even know the existence of TCC Database(s) on purpose, that's what `tccd` is for: since allowing applications to directly modify the TCC Database(s) would be risky, `tccd` (a privileged binary which *is* able to modify the TCC Database(s)) modifies the TCC Database(s) after performing checks and validations (such as requiring the user to approve granting the application permissions to a certain service before it modifies the database accordingly), meaning that applications cannot arbitrarily modify the databases to grant themselves permissions.

7 What about the alert?

Recognizable to just about anyone who's used an iPhone is the alert which pops up whenever an application requests access to a TCC Service (even if the person doesn't know what TCC is):


"ExampleApplication" Would Like to Access the Camera

ExampleApplication would like access to your Camera to take photos!

Figure 2: Example of an alert requesting access to a TCC Service

While presenting an alert is like the ideal solution to obtain user permission, there comes a somewhat difficult issue to solve with this method: alerts prompted by an application are owned by that application, meaning that they'd be able to control those alerts in any way possible, including, in this case, sending fake presses to the "Allow" button to fool the system into thinking the user willingly granted access to the application for the requested service.

In order to circumvent this issue, TCC (specifically, `tccd`) communicates with the System to have the *System present the alert*, meaning that the alert is within control of the System, not the the requesting application, the implementation details are mostly the same on all processes with one slight difference.

on iOS: `tccd` establishes a connection with *SpringBoard*, the subsystem on iOS which manages the Home Screen, Lock Screen, App Switcher, Control Center, and much much more. `tccd` does this by using `CFUserNotification`, a C API (though not exposed in the iPhone SDK) for non-UI programs (such as `tccd`) to display alerts.

on macOS: `tccd` establishes a connection the application responsible for Notification Center UI [2] (not SpringBoard, as it doesn't exist on macOS).

Once `tccd` establishes the connection with the process that'll display the alert, it sends a couple of properties to the process, such as:

- Title of the alert (*Application X* would like to access *Service*)
- Body text of the alert (usually this is the application's *usage description* for the service, discussed earlier in this paper)
- Buttons for user choice: for services where the user can specify *which* data they'd like the application to access (such as Photos in iOS 14+), there'll be 3 buttons (Allow access to all, Allow access to specified data, Don't allow), for all other services, there'll be just 2 choices, allow/don't allow

Once `tccd` receives a response back from the process displaying the alert as to whether the user granted or denied the application access to the requested service, it updates the appropriate TCC Database, and lets the requesting application know whether they were granted or denied access.

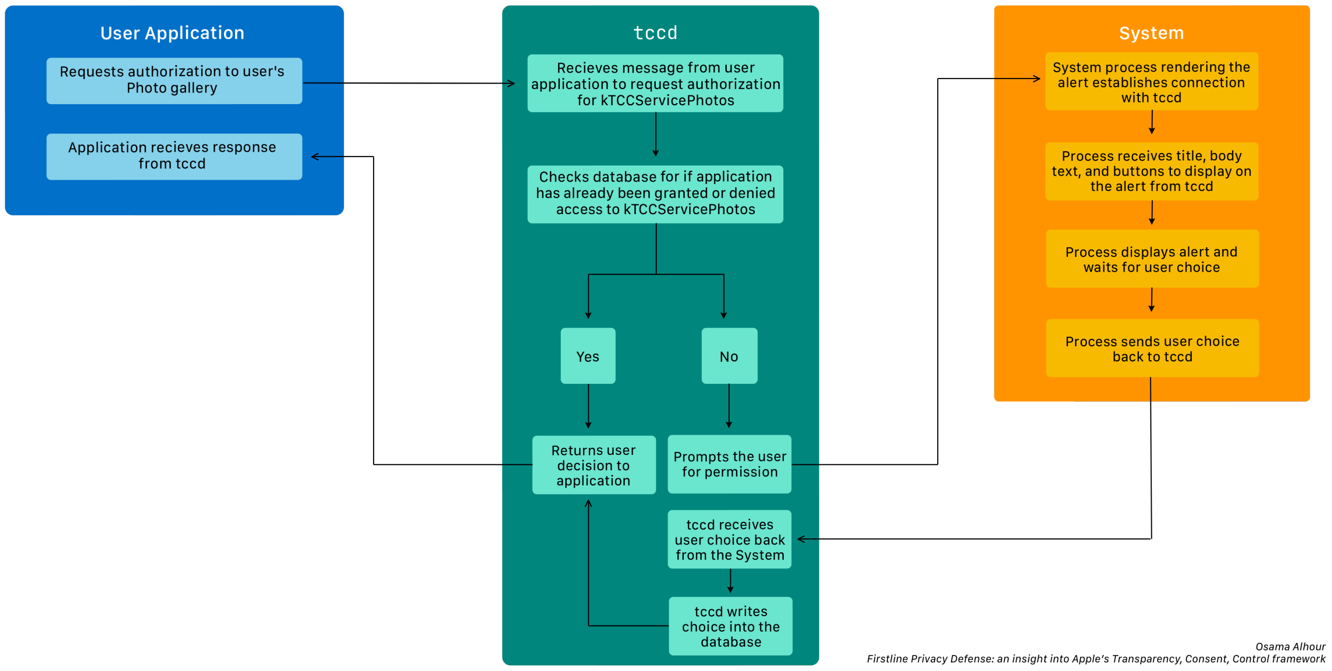

User Application
Requests authorization to user's Photo gallery
Application recieves response from tccd
tccd
Recieves message from user application to request authorization for kTCCServicePhotos
Checks database for if application has already been granted or denied access to kTCCServicePhotos
Yes
No
Returns user decision to application
Prompts the user for permission
tccd receives user choice back from the System
tccd writes choice into the database
System
System process rendering the alert establishes connection with tccd
Process receives title, body text, and buttons to display on the alert from tccd
Process displays alert and waits for user choice
Process sends user choice back to tccd
Osama Alhour
Firstline Privacy Defense: an insight into Apple's Transparency, Consent, Control framework

Figure 3: Diagram demonstrating the flow when an application requests access to a TCC Service.

8 User Authorization for thee, but not for me

Discussed in Chapters 2 and 7 of this paper, an extremely important part of TCC, and what you could argue is the entire point of it, is gatekeeping services behind user permissions, where the user has to *explicitly* authorize the application to use that service when prompted by an alert for it.

Despite this, though, some apps have the luxury of being able to use TCC services *without* prompting the user! Not you, though.

What's going on here? Well, the applications which can access TCC Services without user permissions are limited only to Apple's own applications, not any third-party applications installed from the App Store, and that's only *if* Apple chooses for those apps not to have to ask the user for permission to those services, some Apple apps still request permissions for certain services (i.e., the Maps application still requests permission for the Location TCC Service), while others don't (i.e., iMessages doesn't need to request permission for the Microphone TCC Service).

So how come? Apple has an intricate system called *Entitlements* where applications can specify extra capabilities that the system should provide to that application, sounds awesome, right? The catch is that Entitlements are limited based on which ones Apple allows you to possess, which is strictly enforced by the System, as having Entitlements which Apple didn't grant you is guaranteed to get your app rejected from the App Store, and even if you sideload the application, the System will refuse to launch it.

Since the applications in this context (ones which can access TCC Services without having to ask the user) are Apple's own applications, Apple can simply grant any Entitlements to their own applications, which is what they do in this case.

We can use built-in tools in macOS to check for the Entitlements of any application, let's check it for iMessages:

```
> codesign --display --entitlements - /System/Applications/Messages
    .app >> iMessageRawEntitlements.plist # Dump the Entitlements
    of iMessages using the codesign tool
> /usr/libexec/PlistBuddy -c print -x iMessageRawEntitlements.plist
     # Print it in readable XML format
<?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>
  ...
  <!-- Lots of Entitlements that we dont care about here... -->
  <key>com.apple.private.tcc.allow</key> <!--  Oh!  -->
  <array>
    <string>kTCCServiceAddressBook</string>
    <string>kTCCServicePhotosAdd </string>
    <string>kTCCServicePhotos</string>
    <string>kTCCServiceMediaLibrary</string>
    <string>kTCCServiceMicrophone</string>
    <string>kTCCServiceCamera</string>
  </array>
  <!-- Also lots of Entitlements that we dont care about... -->
  ...
</dict>
</plist>
```

`com.apple.private.tcc.allow`? Yep, it turns out that when an application requests access to a certain service, `tccd` will check it's Entitlements to see if that application has the `com.apple.private.tcc.allow` entitlement, an array of TCC Service names that the app should be granted access to that service without prompting the user, no questions asked. This is how Messages has access to a user's Contacts, Photos, Microphone, and Camera, without requesting access to any of these services from the user.

We can confirm this by reversing `tccd` and reading it's disassembled code:

```
    /* get string of requested service */
    service = objc_msgSend(v5, "service");

    /* use helper class to check if requesting application has com.
    apple.private.tcc.allow array in entitlements
and if it includes the requested service
*/
    v74 = (unsigned int)objc_msgSend(
                            helperClassInstance,
                            "hasEntitlement:containsService:options:"
    ,
                            CFSTR("com.apple.private.tcc.allow"),
                            service,
                            1LL);
    if ( v74 ) {
      os_log(
          OS_LOG_TYPE_DEFAULT,
          "Granting %{public}@ access to %{public}@ via entitlement
     'com.apple.private.tcc.allow'",
          application_name,
      service);
    /* rest of code is to grant access to the application for the
    requested service */
     }
```

9 ...But who does the checking?

So far we've discussed how/where TCC keeps records, how it prompts the user for permission securely, etc etc. But TCC is just the subsystem which keeps tracks of *permissions*, something else has to be providing the data or hardware features which are gated by TCC, so how do they all connect to each other?

The simple answer is... daemons! The classic model is that there 3 players involved when an application accesses a TCC Service:
- The user application
- `tccd`
- A daemon providing the data

For example, user applications use the Contacts framework, which connects to the `contactsd` daemon, a privileged binary which has access to the Contacts database. `contactsd` uses TCC in order to check whether or not the user application requesting contacts has authorization for Contacts before providing the application with a list of contacts. Most other daemons that provide TCC Services follow a similar pattern.

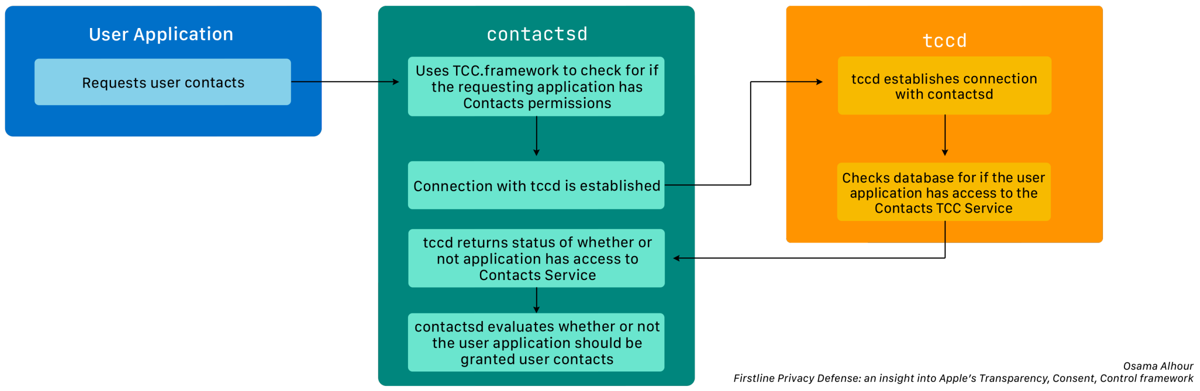

User Application
Requests user contacts
contactsd
Uses TCC.framework to check for if the requesting application has Contacts permissions
Connection with tccd is established
tccd returns status of whether or not application has access to Contacts Service
contactsd evaluates whether or not the user application should be granted user contacts
tccd
tccd establishes connection with contactsd
Checks database for if the user application has access to the Contacts TCC Service
Osama Alhour
Firstline Privacy Defense: an insight into Apple's Transparency, Consent, Control framework

Figure 4: Demonstration of a data-providing daemon using TCC.framework to check if the client application has appropriate permissions

References

[1] Apple, Inc (2021) *Delivering an Enhanced Privacy Experience in Your Photos App* `https://developer.apple.com/documentation/photokit/delivering_an_enhanced_privacy_experience_in_your_photos_app`

[2] Jonathan Levin (2016) *MacOS and iOS Internals, Volume III: Security & Insecurity* `https://newosxbook.com/home.html`

[3] Keith Johnson (2021) *A deep dive into macOS TCC.db* `https://www.rainforestqa.com/blog/macos-tcc-db-deep-dive`