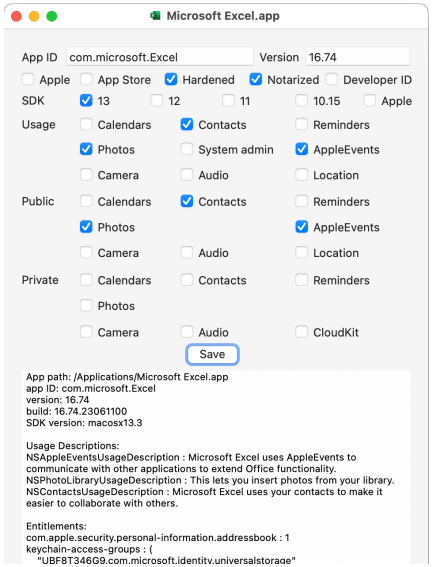
Start



Taccy tells you all about an app's abilities to access private data that's protected by Mojave and later, and provides more general information, including signature validity and whether an app is notarized or was obtained from the App Store. It also checks whether Installer packages are correctly signed and notarized, and makes it easy to check macOS installer apps.

Its log browser helps resolve problems with TCC's* privacy control. This can show you what's going wrong when an app or command runs into problems accessing protected data or services. To use this, you *must* be running as an admin user.

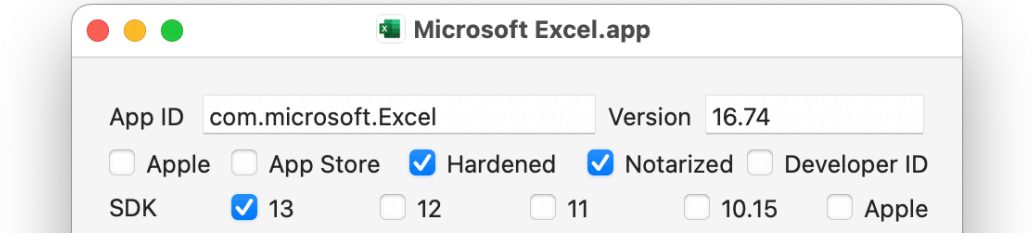
To examine an app or Installer package, simply drag and drop it onto Taccy's icon, whether it's running or not. You can also use the **Open…** command in Taccy's **File** menu.

* TCC = Transparency Consent and Control, the sub-system in macOS responsible for managing privacy.

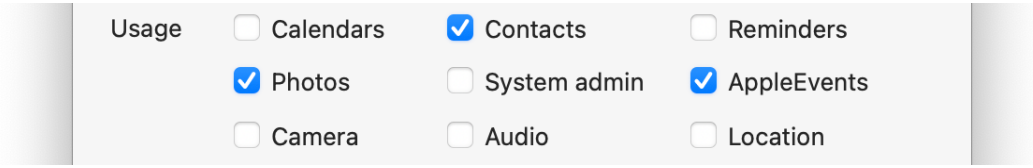
➜ Results (apps) ➜ Open log ➜ Test TCC ➜ Log entries ➜ Technical Information
➜ Packages ➜ macOS installer apps

Results (apps)

Taccy's main window contains two text boxes, many checkboxes (which you cannot change), and a large text scroller at the bottom. These show:



- **App ID**: the standard bundle identifier of the app.
- **Version**: the version number given in the app's Info.plist file.
- The **class of app**: signed by Apple, delivered by the App Store, Hardened, Notarized (shown in Mojave only), or simply signed using a Developer ID.
- **SDK**: whether the app is built against the 13.0, 12.0, 11.0, or 10.15 SDK, or an Apple internal version of Xcode and its SDK.

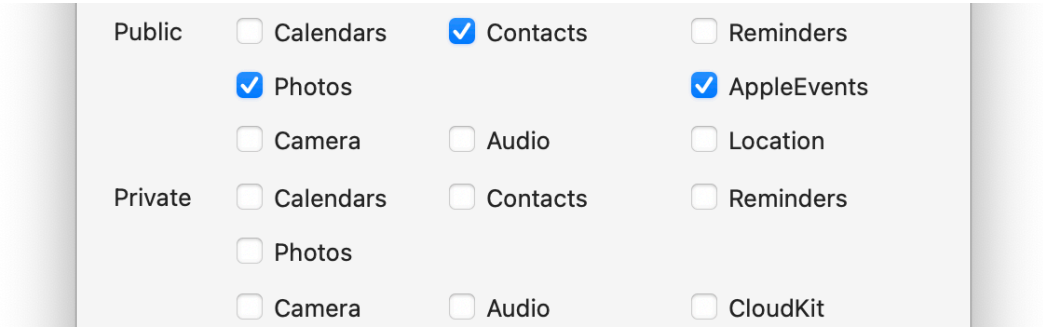


- **Usage**: which classes of protected data are given usage information strings in the app's Info.plist file.

➜ Results (continued) ➜ Packages

➜ Start

Results (continued)



- **Public**: which entitlements to access protected data are granted in its signature using standard public access flags, such as com.apple.security.personal-information.addressbook. These are accessible to all developers, and extensively used by App Store and other sandboxed apps.

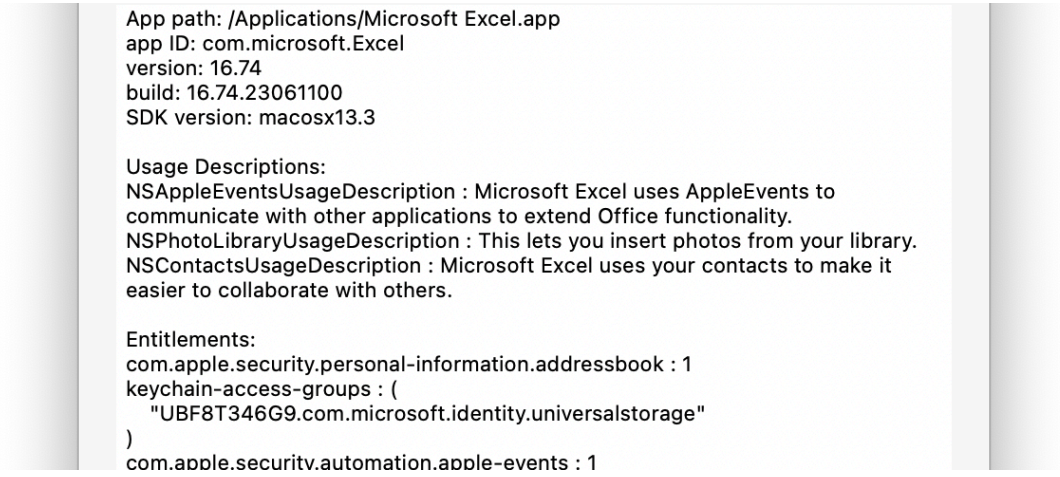
- **Private**: which entitlements to access protected data are granted in its signature, using Apple's private com.apple.private.tcc.allow flags. Only apps built and signed by Apple should be granted these.

Click on the **Save** button to save the contents of the scrolling text box to a text file.

➜ Results (concluded)

➜ Start

Results (concluded)

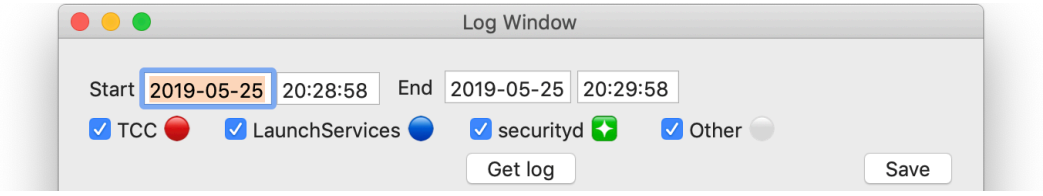


The scrolling text box at the bottom shows the raw information on which the upper textboxes and checkboxes is based, including the entire entitlements listing. To enlarge the text press ⌘+ or to shrink it press ⌘–.

When an app has been built using the 10.14 SDK or later, it must contain usage information strings for any protected data it might want to access, or the user must add it to the **Full Disk Access** list in the Privacy settings. Providing usage information should ensure that TCC will consider attempts to access those classes of protected data are legitimate, and the user will be prompted to consent even though they haven't added it to the **Full Disk Access** list.

➜ Open log ➜ Technical information

Open log



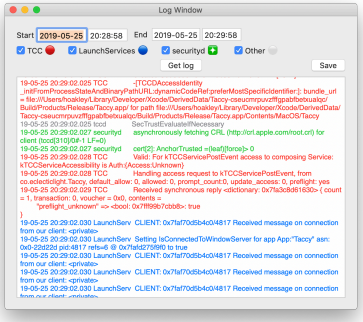
Open a Log window at any time using the **Open Log Window** command in the **Window** menu. *You must be running as a admin user to obtain log extracts.* If you aren't, Taccy doesn't offer this command. Taccy provides three rows of controls at the top of this window:

- **Start** provides the start date (as `YYYY-MM-DD`) and local time (as `HH:MM:SS`) for the start of the log extract created when you click on the **Get log** button.
- **End** provides the end date and local time (same formats as **Start**) for the end of the log extract created when you click on the **Get log** button.
- The **TCC**, **LaunchServices**, **securityd**, and **Other** checkboxes set whether log entries from those subsystems are shown in the text output area. These do *not* determine whether those log entries are collected, only whether they are displayed and saved to file. Alongside each label is the colour in which they are displayed below.
- **Get log** obtains an extract from the log for the set period, and displays its messages in the lower text output view. Getting the log extract can take a significant period of time. During this wait, Taccy keeps that activity in the background and displays a 'busy spinner' next to this button.
- **Save** opens a file save dialog in which you should select a file into which to save the current contents of the text output in the large scrolling text area. These are saved in Rich Text (RTF) format to preserve their colour coding.

➜ Open log (concluded)

➜ Start

Open log (concluded)



Below the buttons is the scrolling text output area. You can copy from that, or save using the **Save** button. To increase the text size press ⌘+ and to shrink it press ⌘–. It also supports a standard **Find** feature, through that command in the **Edit** menu. Whenever you tick or untick one of the checkboxes above, the whole contents are refreshed, and any changes are discarded, including those to font and size. Tip: if the log is very full, before changing text size turn off 3 of the 4 checkboxes to make the operation quicker.

Default dates and times are set to capture logs for a period of one minute prior to opening the log window.
Four types of log message are displayed:

- from the `com.apple.TCC` subsystem, displayed in black/white, controlled by the **TCC** checkbox,
- from the `com.apple.launchservices` subsystem, displayed in red, controlled by the **LaunchServices** checkbox,
- from the `com.apple.securityd` subsystem, displayed in green, controlled by the **securityd** checkbox,
- those from `tccd` and `sandboxd` processes, and the kernel, displayed in blue, and controlled by the **Other** checkbox.

Those checkboxes *don't* affect the log entries collected by or stored in Taccy, only the display of those types of log message. They also determine what is saved when you save the output view to an RTF file: types which are not displayed are not saved in the file.

Each log entry is displayed in the following fixed format:
`YY-MM-DD HH:MM:SS.sss [subsystem] eventMessage`
where `[subsystem]` is the name of the subsystem, e.g. TCC, the process (either tccd or sandbox), or [other].

Open as many Log windows as you wish. Close them using the normal red Close button on that window.

➜ Test TCC ➜ Log entries

➜ Start

Test TCC

Tests of TCC and log capture are performed manually. This is because the manner in which an app or command is opened/run has great bearing on the way in which it is handled by TCC.

To run a test using the Log window, open Taccy first, but don't open a Log window yet. Watch the system clock in your menu bar, and perform whatever actions you want to test using the app or command tool you are investigating. Do these at times which you can remember simply, over a period no more than 45 seconds, noting the second count for each action you want to test.

Once you have completed the test, open a new Log window. This will have the 60 seconds prior to opening set in its **Start** and **End** boxes ready for you to use. Ensure that the **Start** time is a few seconds before the clock time of the first action, the **End** time is at least a few seconds after the clock time of the last action, and the time difference between the **Start** and **End** is 10 seconds or longer, then click on the **Get log** button to view the log during the test process(es).

➜ Log entries

Log entries

Log excerpts are rich in information which can reveal a great deal about how the privacy system and other parts of macOS handle the actions of an app or command. Here are some key entries which you are likely to come across. In each case, they relate to my app xattred, which has been notarized.

```
[LaunchServ] LaunchedApplication: "/Applications/xattred.app/Contents/MacOS/xattred", psn=[ 0x0/0xd51d51]
```
The app xattred.app has been launched by LaunchServices, which is the start of all the TCC checks.

```
[TCC] AttributionChain: ACC:{ID: co.eclecticlight.xattred, PID[28287], auid: 501, euid: 501, binary path:
'/Applications/xattred.app/Contents/MacOS/xattred'}, REQ:{ID: com.apple.WindowServer, PID[200], auid: 88,
euid: 88, binary path: '/System/Library/PrivateFrameworks/SkyLight.framework/Versions/A/Resources/
WindowServer'}
```
TCC has established an Attribution Chain for the request. This gives the process which is making the request, here WindowServer, and the app which is responsible for satisfying TCC's requirements, xattred.app. This is particularly important with command tools and other processes which have no GUI to interact with the user.

```
[tccd] SecTrustEvaluateIfNecessary
```
This marks a call to evaluate security certificates if required, in this case initiated by tccd.

```
[securityd] asynchronously fetching CRL (http://crl.apple.com/root.crl) for client (tccd[310]/0#-1 LF=0)
```
The security certificate is being checked by securityd.

```
[TCC] /Applications/xattred.app/Contents/MacOS/xattred (offset 0) linked against SDK version 0xa0e00
```
TCC has checked the app, and discovered that it was built against macOS 10.14 SDK. It therefore applies the latest and most stringent privacy rules.

➜ Log entries (concluded)

➜ Start

Log entries (concluded)

```
[TCC] Prompting policy for hardened runtime; service: kTCCServiceAppleEvents requires entitlement
com.apple.security.automation.apple-events but it is missing for ACC:{ID:
co.eclecticlight.xattred, PID[28287], auid: 501, euid: 501, binary path: '/Applications/
xattred.app/Contents/MacOS/xattred'}, REQ:{ID: com.apple.appleeventsd, PID[68], auid: 55, euid:
55, binary path: '/System/Library/CoreServices/appleeventsd'}
```

As a notarized app, the runtime is 'hardened'. This requires it to have an entitlement in order to access AppleEvents automation services. The app doesn't have that entitlement, so cannot use those services.

```
[TCC] matchesCodeRequirementData: SecStaticCodeCheckValidity() static code (0x7fae018388d0) from
co.eclecticlight.xattred : anchor apple generic and identifier "co.eclecticlight.xattred" and (certificate
leaf[field.1.2.840.113635.100.6.1.9] /* exists */ or certificate 1[field.1.2.840.113635.100.6.2.6] /*
exists */ and certificate leaf[field.1.2.840.113635.100.6.1.13] /* exists */ and certificate
leaf[subject.OU] = QWY4LRW926); result: 0
```

TCC has checked the code requirement for the app, and it has passed those checks.

```
[TCC] Handling access request to kTCCServiceSystemPolicyAllFiles, from co.eclecticlight.xattred,
default_allow: 0, allowed: 0, prompt_count:0, update_access: 0, preflight: no
```

TCC is here checking whether the app has Full Disk Access.

```
[TCC] Handling access request to kTCCServiceCalendar, from co.eclecticlight.xattred, default_allow: 0,
allowed: 1, prompt_count:1, update_access: 0, preflight: no
```

This test included opening a protected file with the user's Calendars folder. This is the access request to TCC to permit that.

```
[TCC] Received synchronous reply <dictionary: 0x7fefe3e12530> { count = 1, transaction: 0, voucher = 0x0,
contents = "result" => <bool: 0x7fffa7a20be8>: true}
```

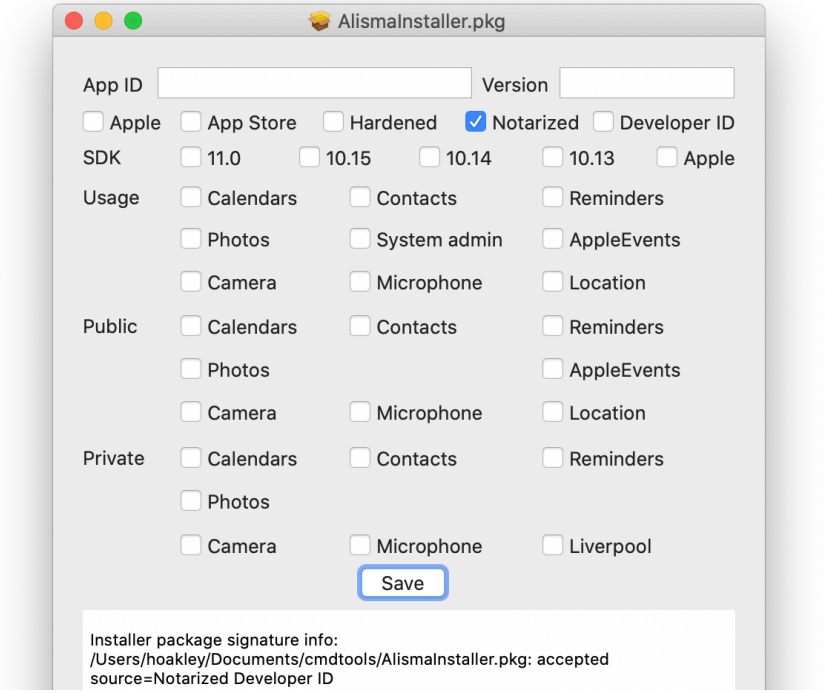
This is a typical successful reply from TCC to a request to access protected folders or services.

```
[sandbox] kTCCServiceCalendar granted by TCC for xattred
```

Apps which run in a sandbox (App Store), or in this case are hardened, will also record the granting of requests to access protected folders or services.

➜ Technical information

Installer packages



When you open an Installer package in Taccy, the app checks its signature, using the calls

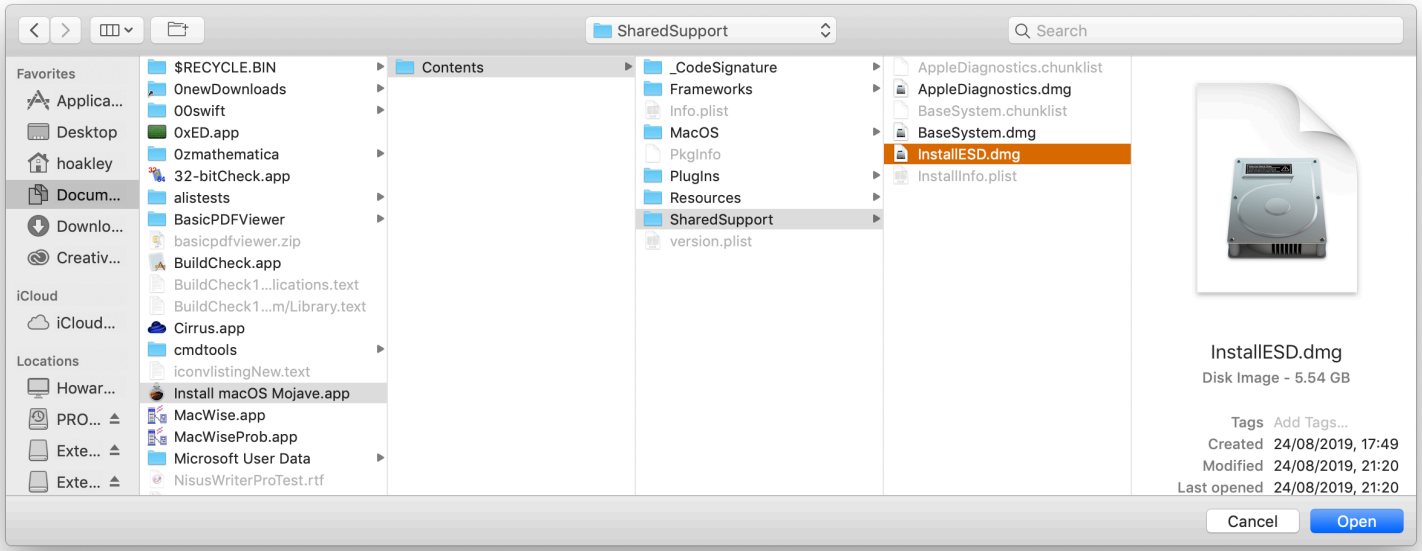
```
spctl -t install -a -vv /path/pkgname.pkg
pkgutil --check-signature /path/pkgname.pkg
```

It then sets the signature checkboxes at the top, and reports the result of that check in the text view at the foot. This includes whether the package has been signed or notarized. Errors are indicated by ⛔ in the report.

➜ Results (apps)

macOS installer apps

macOS updates are usually delivered as Installer packages, which can be checked by dropping them onto Taccy, or using the **Open** menu command. Full macOS installers are normally apps, and simply checking them as an app isn't going to tell you whether they still work. To do that, you need to check one of the Installer packages embedded with them. For that, you'll need to locate one of the disk images inside the installer app, mount it, then check a package inside it. Taccy makes this simpler for you.

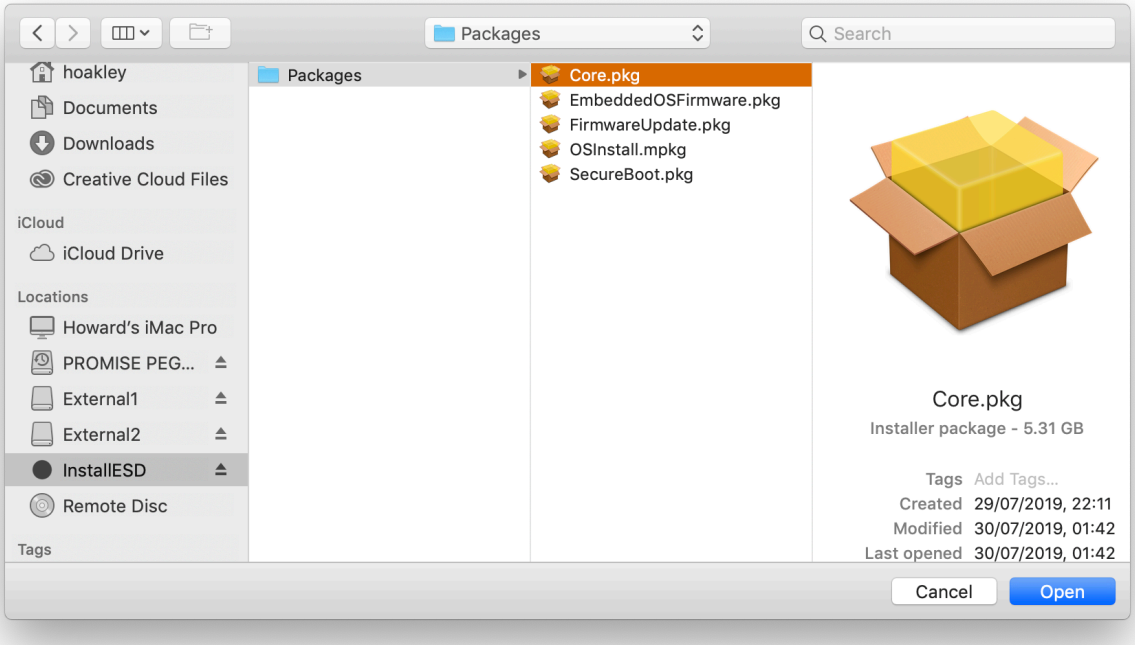


Use the **Mount Disk Image** command in Taccy's **File** menu first. In the Open File dialog, open the installer and work your way down to the **SharedSupport** folder containing disk images. Select the largest of those, often called InstallESD.dmg, and click on the **Mount** button to mount that disk image in the Finder.

➜ macOS installer apps (concluded) ➜ Installer packages

➜ Start

macOS installer apps (concluded)



When the disk image has mounted in the Finder, use Taccy's **Open** command in the **File** menu to open that volume, and work your way down to the folder of packages. Select one of these (the largest is ideal), and click on the **Open** button for Taccy to check it. You can also locate the package in the Finder instead, and drag and drop it onto Taccy, if you prefer.

➜ Installer packages

➜ Start

Technical information

Taccy derives its information from two sources in each app: the Info.plist file, which contains the App ID, version, SDK information, and any usage strings which are declared; the app signature, which contains all the entitlements. The latter are extracted using the command

```
/usr/bin/codesign --display --entitlements :- /path/appname.app
```

which returns a property list, from which individual entitlements are obtained.

`codesign` works well with more recent SDKs and signatures, but older ones are less reliable. In some cases, Taccy will be unable to parse the property list fully, and may dump that to the scrolling text box without listing its dictionary in full. Older apps don't seem to return a property list at all, and an error will be reported instead.

Notarization/class and the code signature are checked using the commands

```
spctl -a -vv /path/appname.app
codesign -dvvv /path/appname.app
```

Because these check signature status with Apple, this takes a little time, returning its results asynchronously. In some cases, it may not return a complete result. Note that the version included in Sierra and High Sierra doesn't know about Notarization: notarized apps are only recognised as such in Mojave and Catalina.

Log extracts are obtained using `log show`, in a command of the form

```
log show --predicate 'subsystem == "com.apple.TCC" OR subsystem == "com.apple.launchservices" OR subsystem
== "com.apple.securityd" OR subsystem == "com.apple.sandbox" OR processImagePath CONTAINS[c] "tccd" OR
processImagePath CONTAINS[c] "sandboxd" OR processID = 0' --style json --info
```

for the time period specified by the **Start** and **End**.

This returns the log entries in JSON format, which are then parsed entry by entry to create a styled attributed string containing all the log entries in sequence.

➜ Change list

Change list

1.15:

- flags updated and rebuilt for Sonoma.

1.14:

- changes initial log checks to detect errors in time formatting
- correct other time formats to allow for 12-hour clock
- improves speed and reduces memory requirement for logs.

1.13:

- adds initial log checks to avoid macOS bugs.

1.12:

- adds support for all 11.x SDKs
- supports changed privacy entitlements and scope.

1.11:

- improves text colours in the log window
- Universal App.

1.10:

- adds support for the 11.0 SDK as well!

1.9:

- adds support for the 10.16 SDK.

1.8:

- fixes a problem in assessing whether an admin user for those users who are members of few groups.

1.7:

- added kernel entries to predicate for log browser
- changed font of log browser to system monospace
- reformatted log browser entries
- now tests whether admin user to decide whether to offer log browser.

1.6:

- added Mount Disk Image menu command
- added busy spinner for signature checks
- added info on installer apps to Help book
- added support for opening old installer packages.

1.5:

- improved signature error reporting
- added pkgutil check for packages.

1.4:
- added support for checking Installer package notarization
- ported to Swift 5.1 and Xcode 11.1.

1.3:
- added codesign and hardening assessment, and improved error reporting.

1.2:
- changed method of obtaining UsageDescriptions to work with Catalina's much-expanded list.

1.1:
- added auto-update support
- added support for changing text size to main and log windows
- fixed crashing bug when trying to obtain list of entitlements
- added build version to list of info obtained
- added support for Catalina SDK.

1.0:
- first full release
- ported to Swift 5 and Xcode 10.2.1
- added code integrity check on open
- added PDF Help book
- improved functionality of scrolling text view.

1.0b7:
- added log browser window.

1.0b6:
- ported to Swift 4.2.1 and Xcode 10.1
- notarized for Mojave.

1.0b5:
- added other classes of app to checkboxes
- added AppleEvents to the Public group of checkboxes
- improved window layout.

1.0b4:
- added Notarized checkbox (which only works in Mojave) and code signing information to text
- build using Xcode 10.0 release.

1.0b3:
- initial release.

19 June 2023.