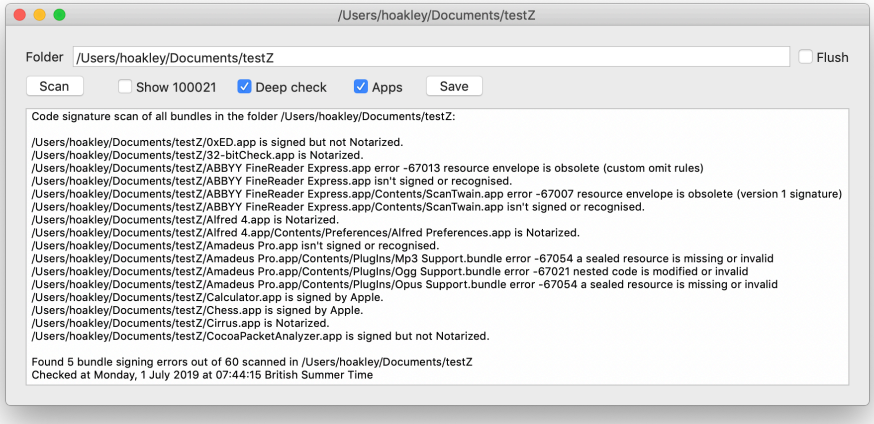
➜ Start

Start



Signet is a free utility for checking the signatures of apps and other bundles in macOS. Normally, these are checked thoroughly just once, when they are first run after being downloaded and installed. If you keep migrating from one Mac to the next, chances are that your current Mac contains a lot of executable code which has no signatures at all, and possibly some for which the signatures have been revoked.

Checking signatures is a good way to discover old software left over from the past which has never been cleared away properly. It can also reveal apps and bundles which, because of their signature weakness or absence, could be exploited or abused. When I was developing Signet, I discovered a Microsoft app whose signature had been revoked, which macOS Mojave was quite happy to open despite that, a lot of properly signed apps which contained unsigned apps inside them, and a great many apps and other bundles (some quite recent) which have no signature at all. Signet can now also check whether an app has been notarized, etc.

➜ Use ➜ Interpretation ➜ Notarization ➜ Updates ➜ Technical information

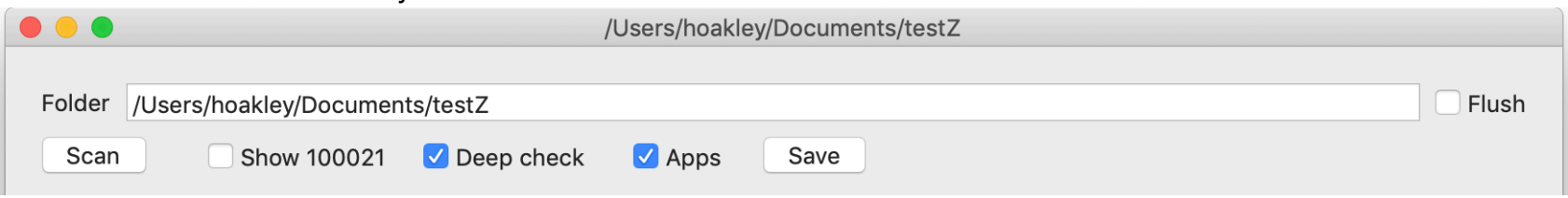
Use

Signet checks bundles, including apps, on any storage accessible from your Mac. These are special types of folder with set contents and structure, and are standard for apps, various types of plug-in, extensions, frameworks, and more. Most contain executable code, which should mean that they are signed, even when nested inside a signed app.

Specify the folder to be searched in any of three ways:

- Type the path to a folder in the box next to the word **Folder**, then click on the **Scan** button, for Signet to search that folder. If that folder does not exist, it will do nothing. You can start this folder path with the special character ~ as shorthand to indicate your Home folder, e.g. ~/Applications is the Applications folder in your Home folder.
- If there's no text in the box next to the word **Folder** when you click on the **Scan** button, Signet displays a normal Open File dialog in which you should select the folder you wish to search, then click on the **Open** button.
- Drag and drop any folder from a Finder window into the box next to the word **Folder**, then click on the **Scan** button to scan that folder.

Normally, when you insert the path to a folder in the Folder textbox, that path is cleared when you perform the next scan. If you want the path to remain there, so that you can modify it, perhaps, uncheck the **Flush** checkbox. When it is ticked, the textbox will be cleared of its contents each time that you scan.



Three scanning options are set using checkboxes in Signet's windows:

- **Show 100021** determines whether the check should also report those bundles which return an error type 100021 indicating that trying to access the bundle failed because of an 'illegal operation on a directory'.
- **Deep check** determines the thoroughness of signature checking. By default, it is ticked, and checks for and reports most if not all errors; if you uncheck it, tests will be quicker, but only detect unsigned bundles, those with revoked certificates, and more significant errors.
- **Apps** determines whether to check if apps are notarized, signed by Apple, delivered by the App Store, etc. See ➜ this page.

➜ Use (concluded)

➜ Start

Use (concluded)

Scans run in the background, displaying a ‘busy’ spinner next to the **Scan** button to indicate that Signet is still working hard to complete the scan. Signet inspects every item within your chosen folder to determine if it is a bundle, and whether it contains executable code. If it contains executable code, Signet asks macOS to validate its signature. Any errors reported are then compiled to form Signet’s output. When it has checked any bundle, it then looks inside that for nested bundles, checking those as appropriate.

Once its search is complete, Signet summarises its results in the lower scrolling text view. These include the total number of executable bundles found, out of the total number of executable bundles found. To save the report, click on the **Save** button and select an appropriate or new file.

When Signet scans bundles, it does so exhaustively, which takes significant time and memory. The checks it performs may require online validation of certificates, which can also slow them down. If you ask Signet to check any folder containing a very large number of bundles, that will be protracted, involve many signature checks, a lot of CPU time, and lots of memory too. Don't be tempted to try it on the root folder of your startup volume, or it could take many hours. With a little patience, it can be used on individual top-level folders such as /System/Library (dull but important), /Library (rich and very important), /Applications (often rich and surprising), and ~/Library (rich and important).

Enlarge text in the scrolling text view using ⌘+ or reduce it using ⌘–. This changes in 1 point increments between 4 and 60 points. Signet can open multiple windows. When you’re done with the app, use the **Quit** command to quit it.

⚠️ Signet doesn’t have the ability to scan any folders containing private data, including your photos, email, address book, etc. Those folders shouldn’t contain executable code, so this app shouldn’t be going anywhere near them. However, if you want to include those protected folders in its scans, add it to the **Full Disk Access** list in the **Privacy** tab of the **Security & Privacy** pane.

There is a known vulnerability in signature checking which means that crafted signatures can sometimes cheat the validation system used by Signet. Signet is not intended to detect malware, but to scan non-malicious apps and bundles for signature errors. If you want to detect malware, use one of the several excellent anti-malware apps such as Malwarebytes.

➜ Interpretation
➜ Notarization
➜ Technical information

Interpretation

Scans performed with the **Show 100021** checkbox ticked are likely to return large numbers of errors of type 100021, kPOSIXErrorEISDIR, an 'illegal operation on a directory'. These appear benign and irrelevant.

Error codes from signature validation are explained as meaningfully as macOS permits. Common ones include:

- -2147409652 CSSMERR_TP_CERT_REVOKED, the certificate has been revoked
- -67007 resource envelope is obsolete (version 1 signature)
- -67008 unsealed contents present in the root directory of an embedded framework
- -67013 resource envelope is obsolete (custom omit rules)
- -67021 nested code is modified or invalid
- -67023 invalid resource directory (directory or signature have been modified)
- -67030 invalid Info.plist (plist or signature have been modified)
- -67054 a sealed resource is missing or invalid
- -67056 code has no resources but signature indicates they must be present
- -67061 invalid signature (code or signature have been modified)
- -67062 code object is not signed at all, which is by far the most common.

Many third-party apps which are distributed independently of the App Store use the Sparkle mechanism to offer and install updates. This hasn't been without its vulnerabilities in the past, and one odd feature about many (but not all) apps which currently use Sparkle is that one app nested within them is unsigned: Autoupdate.app or finish_installation.app, which actually runs the update process. This is normally found in the path within the app's bundle of /Frameworks/Sparkle.framework/Versions/A/Resources/Autoupdate.app. Many older apps such as Apple's Aperture and the Bento database used now-obsolete signature formats, and return errors of -67013 “resource envelope is obsolete (custom omit rules)”.

Even current App Store apps may have signature errors in nested components: for example, I have an audio app purchased from the App Store which contains three plug-ins in its app bundle which fail Signet's strict validation, one of which reports that “nested code is modified or invalid”.

Many accessory apps from major vendors such as Adobe and Microsoft are unsigned or return signature errors. For example, Adobe Acrobat 2015 contains two nested apps which lack any signature, and no less than 12 of the support apps and bundles for old Adobe products which are still installed on my Mac have had their certificates revoked.

➜ Interpretation (concluded)

➜ Start

Interpretation (concluded)

Current Microsoft Office apps (16.20) contain large numbers of proofing tools which contain executable code, but are completely unsigned.

Very many frameworks and plug-ins are unsigned or return signing errors. Some of the most serious include Apple's Compressor.framework and Qmaster.framework stored in /Library/Frameworks, which use an ‘obsolete resource envelope’ from an old signature format, two StuffIt frameworks, Microsoft's SilverLight Internet Plug-In which is completely unsigned, the Java Virtual Machines stored in /Library/Java, in which even the JDK 10.0.2 and its preference pane are unsigned. Most older QuickTime components are reported with signature errors, with many lacking any signatures.

So what should you do with a bundle for which errors are reported? That depends on what the bundle is for and what the error is. Some reputable software developers consider that what Signet does is completely worthless. They argue that signatures are only there for Gatekeeper, and as Gatekeeper is only there to check apps when they are first run, it doesn’t matter once the app has passed that check. Others consider that checking signatures is “security theatre”, and unless those checks were accompanied by databases containing ‘true’ signature references, they are worthless.

➜ Notarization
➜ Technical information

➜ Start

Scan apps for notarization

Mojave 10.14.5 and Catalina 10.15 introduce new requirements for certain apps and other software to be notarized. To help you prepare for this, Signet's **Apps** option checks whether apps with the bundle extension .app have undergone notarization, or are exempt because they're signed by Apple or delivered from the App Store. This uses the command tool `spctl` in the call

```
spctl –a –v appPath
```

which *adds greatly to the time required for scanning.* Although Signet can perform this on a large Applications folder, that takes a very large number of calls to `spctl`, a lot of memory, and tens of minutes to complete. If possible, scan smaller folders with no more than 50 apps at a time when the **Apps** option is ticked.

For each app discovered during the scan, Signet parses the response from `spctl` and classifies the app into one of the following:

- **is Notarized**, when the response states that the app has been notarized correctly;
- **is signed by Apple**, when the signature is Apple's;
- **is an App Store app**, when the app has been supplied from the App Store and is exempt notarization;
- **is signed but not Notarized**, when a signature is found but the app hasn't been notarized by Apple;
- **isn't signed or recognised**, in all other cases.

Currently, Signet checks notarization on all nested apps, but *not* on other code bundles or standalone code such as Mach-O binaries.

➜ Technical information

Updates

Whenever you open Signet, it may check to see if an update is available. This *doesn't* use the popular Sparkle mechanism for updating in place, but works as detailed here.

Once Signet has successfully completed its integrity check, it checks whether update checking has been turned off in its preferences file. If that has, it abandons any attempt to check for updates. If checking is allowed, it then checks when it last checked for updates. If that was more than 12 hours ago, it continues to perform the check. It then connects to my GitHub server, from where it downloads a list of current versions of my apps. It doesn't upload any data to the GitHub server at all, and no statistics beyond GitHub normal connection figures are collected either: no personal identifiers are recorded.

If there is an update available, Signet then checks that its location is on my WordPress blog, and posts a dialog which invites you to download the update.

If you click on the **Download** button, it then points your default browser at that update, which should trigger the update to be downloaded to your normal downloads folder. The update is received as a regular Zip archive, and is exactly the same as you would download from the Downloads page. It also carries a quarantine flag, so that when you unZip it and install the app inside, it undergoes normal first run 'Gatekeeper' security checks. If you click on the **Ignore** button, Signet won't remind you about it again for another 12 hours.

An additional item at the end of the **Help** menu explains the update status. If no update check is performed, or the check fails, the last item reads **Update not checked**. If the check is performed and update information is obtained, even when no update is available or you decline to download it, that menu item reads **Checked for update** and is ticked (but still disabled).

You can customise this behaviour by changing Signet's preferences. The keys to use are:

- `noUpdateCheck`, a Boolean. When set to `true`, this disables all update checking. Default is `false`.
- `updateCheckInt`, a real number (Double). When set to a value greater than 1.0, the minimum time interval between checks, in seconds. Default is 43200, which is 12 hours. If you set it to any value less than 1, Signet will reset it automatically to that default.

To change either of these, use a Terminal command of the form

```
defaults write co.eclecticlight.Signet updateCheckInt '10'
```

which works properly through the preferences server `cfprefsd`.

Technical information

Signet performs a deep traversal of all items within the chosen path. It checks each URL within the directory to determine whether it is a Finder alias; if it is, it does not do anything further, but if it isn't a Finder alias, it then checks whether it is an executable bundle, i.e. a bundle containing executable code. Errors occurring at this stage are included in Signet's listing only when the **Show 100021** checkbox is ticked/checked.

If it is an executable bundle, Signet treats the bundle as containing 'static code' and asks macOS to perform a validation on it according to the **Deep check** setting. If that is ticked (default), the checks are performed using `SecCSFlags` of `kSecCSStrictValidate`; if that checkbox is unchecked, then `SecCSFlags` are set to `kSecCSBasicValidateOnly`.

There are circumstances in which even strict checking may miss a signature problem, but those are most unlikely to occur except in malware deliberately trying to cheat signature validation. As Signet is not intended for use to detect malware, the speed gain in not implementing a full strict validation is preferred over that completeness.

The two depth settings do not affect the depth of scanning for bundles in any way. Viewed in the unified log, such checks typically appear as:

```
07:26:00.808115 Unable to parse ticket.
07:26:00.808116 error registering ticket: -1
07:26:00.808141 com.apple.securityd MacOS error: -1
07:26:00.808200 com.apple.securityd Error registering stapled ticket: [hex]
07:26:00.808635 SecTrustEvaluateIfNecessary
07:26:00.808944 com.apple.securityd cert[2]: WeakSignature =(leaf)[]> 0
07:26:00.808951 com.apple.securityd cert[2]: WeakSignature =(leaf)[]> 0
07:26:00.809584 com.apple.securityd cert[2]: WeakSignature =(leaf)[]> 0
07:26:00.809588 com.apple.securityd cert[2]: MissingIntermediate =(leaf)[force]> 0
07:26:00.809593 com.apple.securityd cert[2]: WeakSignature =(leaf)[]> 0
07:26:00.809603 com.apple.securityd cert[2]: WeakSignature =(path)[]> 0
07:26:00.809619 com.apple.securityd cert[2]: BlackListedKey =(path)[force]> 0
07:26:00.809766 com.apple.securityd Trust evaluate failure: [root BlackListedKey      MissingIntermediate WeakSignature]
07:26:00.810094 SecStaticCode: verification failed (trust result 6, error    -2147409652)
07:26:00.810098 com.apple.securityd MacOS error: -2147409652
07:26:00.810134 com.apple.securityd MacOS error: -2147409652
```

which is then reported by Signet as:

```
/Users/hoakley/Documents/test2/Microsoft Messenger.app error -2147409652 CSSMERR_TP_CERT_REVOKED
```

At the end of each report, Signet gives the total number of bundles which it checked, and the number of those for which signature validation return an error.

➜ Change list

Change list

Signet 1.3:

- Universal App with support for Big Sur.

Signet 1.2:

- added feature to scan apps for notarization status.

Signet 1.1:

- added auto-checking for updates
- added commands to change text font size
- conforms to build numbering conventions.

Signet 1.0:

- added code signature integrity checking when opening
- added support link
- ported to Swift 5 and Xcode 10.2.1
- tweaked settings on text views to improve functionality.

Signet 1.0b2:

- added less deep check option
- hopefully built for compatibility with High Sierra
- improving timing of busy spinner when scanning.

Signet 1.0b1:

- initial release.

1 September 2020.