MacTEX-2019, notification, and hardened runtimes

Richard Koch

Abstract

MacTEX installs everything needed to run TEX on a Macintosh, including TEX Live, Ghostscript, and four GUI applications: TeXShop, TEX Live Utility, LATEXiT, and BibDesk. In macOS 10.15, Catalina, Apple requires that install packages be notarized, and all command line and GUI applications in such a package must be signed and adopt a hardened runtime. I'll explain what this means and how it was accomplished.

MacTEX 2019

1 Recent changes

For many years, MacTEX supported macOS 10.5 (Leopard) and higher, on both PowerPC and Intel processors. Starting in 2017, we decided to limit support to those systems for which Apple still provides security updates. Consequently, we support the three latest systems; in 2019 we support Sierra, High Sierra, and Mojave (that is, 10.12 and higher). Each fall, Apple introduces a new system and we also support that. Thus MacTEX-2019 will support Catalina when that is released this fall.

Mojca Miklavec compiles Mac binaries for older systems; in 2019 she supports Snow Leopard (10.6) and higher. TEX Live contains both our binaries and Miklavec's binaries. Our web pages (`tug.org/mactex`) explain how to install TEX Live using either the MacTEX installer or the standard Unix install script (`install-tl`), so users with older systems can update using the Unix install script. Both methods produce exactly the same TEX Live in the end.

2 Security

I retired from the University of Oregon in 2002. In that year, freshmen arriving at the University discovered a CD and instruction sheet taped over the ethernet jacks in their dorm rooms. The sheet said

```
Warning: You must install the virus checker on
this CD before connecting your computer to the
ethernet. If you fail to follow this instruction,
you will lose ethernet privileges in this room.
```

The note ended with one more sentence:

```
Macintosh users can ignore this message.
```

But that was 2002. This April, I got the following:

```
From: koch@math.uoregon.edu
Date: April 4, 2019
To: koch@math.uoregon.edu

Hey! I compromised your account and gained full
access to it. I just sent this email from your
account. You visited an adult website and got
infected. This gave me access to all of your
contacts, browsing history, your passwords,
your webcam, and even your microphone.

I noticed you were trying to please yourself by
watching one of those nasty videos, well my son,
I recorded your actions ... (thanks to your
webcam) and even recorded your screen (the video
you were watching). Now, if you do nothing, then
I will send this video to all of your email,
social media and messenger contacts. You have
the option to prevent me from doing all of this.
All you need to do is to make the transfer of
\$958 to my bitcoin address ...
```

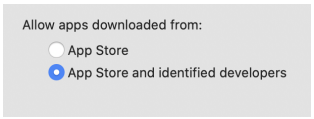
3 Lessons

- The Macintosh is built on top of Unix. Unix has strong protection against *other irresponsible* users. Like most companies, Apple has security engineers patching kernel and system bugs as they are found.
- But Macs are generally used by one person, and the remaining problem is to protect that person against himself or herself. If my Mac is attacked, I'm not worried that the criminal will become root. I'm worried that he will activate my camera, read my mail, find my contact list, or turn on my microphone.
- For several years, Apple has provided a (mandatory) solution for applications in the App Store. It is known as *sandboxing*. A sandboxed application cannot interact with other programs; it runs in its own sandbox.
- In Catalina (and also to some extent in Mojave) Apple provides a different kind of security protection for other programs. Unlike sandboxing, the new security is carefully tuned to allow any program to run as usual. Here's how it works.

4 Signing

This step was introduced in 2012. Apple Developers can *sign* their applications and their install packages. When software is downloaded from the Internet, the system checks that the software has not been modified since it was signed, and that the signature is from a known developer. It refuses to run software that doesn't pass. Otherwise it sets a Finder bit to disable future checks and runs the software. A

control panel in Apple's System Preferences controls this behavior:



Signing requires developer status from Apple, which costs $100 a year. TeXShop and MacTEX have always been signed.

Apple issues *two* developer signing certificates, one for applications and one for install packages. Signing applications is done in XCode as part of the build process. A command line binary signs install packages.

Tricks explained on the Internet allow users to disable the signing requirement and install any program. At this year's WWDC, Apple said that such tricks would *always* be available.

5 Notarization

This spring, Apple added notarization. This works like signing; both applications and install packages can be notarized. Once software is signed and just before release, it is sent to Apple. There it is checked for viruses (no human hands touch the software). Checking takes around 15 minutes. If the software passes the test, a "certificate" is mailed back and "stapled" to the software. In Catalina, software downloaded from the Internet must be both signed and notarized before it can run.

Previously, software was only tested once to make sure it was not modified. Now these tests will be rerun periodically. The details are somewhat vague (to me), so don't ask.

6 Hardened runtimes

Signing and notarization are small potatoes. The big security step in Catalina is the requirement that all applications and command line programs in a notarized install package must be signed and timestamped, and must adopt a Hardened Runtime. All of this is new. The MacTEX install package has been signed since 2012, but the individual TEX binaries are not signed. And while TeXShop is signed, the remaining applications TEX Live Utility, LATEXiT, and BibDesk are not signed. The kicker, however, is that these applications *and all command line apps* must adopt a hardened runtime. What is that?

Apple has a list of 13 dangerous operations a program might try to perform. I'll give the full list later, but among the items are these: accessing the camera, accessing the microphone, accessing location information, accessing the address book, accessing the user's calendars, accessing photos, sending Apple events to other applications, executing JIT-compiled code, loading third party libraries not by Apple. If an application adopts a hardened runtime, it is not allowed to perform any of these operations.

However, for each of the 13 dangerous operations, a developer can claim an *entitlement*. I have always dreamed of a TEX editor attached to a camera; to make a commutative diagram, draw it and take a picture and the editor converts the drawing into TEX. The author of such an editor would file an entitlement for the camera operation.

Nobody at Apple checks the entitlement list; there is no "approval process". A developer can claim all 13 entitlements and then the hardened runtime has no effect.

So calm down that case of paranoia. Apple isn't restricting developers. It is providing a tool to help open source developers improve security.

6.1 Dealing with command line programs

Command line programs can adopt a hardened runtime without recompiling. The command below does this for the `xz` binary used by `tlmgr`. The `--force` option says to replace any previous signing by the new one, and `--options=runtime` says to adopt a hardened runtime with no exceptions.

```
codesign \
 -s "Developer ID Application: Richard Koch" \
  --force --timestamp  --options=runtime xz
```

To claim exceptions for a command line program, add a flag `--entitlements=TUG.entitlement` to the previous call, where `TUG.entitlement` can be any name and is a short XML file. The example `TUG.entitlement` here allows linking with third party libraries. (One long line has been broken for *TUGboat* with a \; it should not be broken in a real file.)

```
<?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.security.cs.\
disable-library-validation</key>
  <true/>
</dict>
</plist>
```

By embedding the `codesign` call in a shell script, it is easy to construct scripts which sign, timestamp, and adopt hardened runtimes for all command line binaries in an install package.

6.2 Case 1: BasicTEX

In addition to the full MacTEX, we provide a smaller install package called BasicTeX, which installs the distribution obtained by using `install-tl` with the "small" scheme. To test the above ideas, I submitted this package unmodified to Apple for notarization. Apple refused to notarize it, but they sent back a detailed and easy-to-read error sheet. The bin directory of BasicTEX has 88 items. Apple ignored symbolic links, scripts, and other files, but had problems with 30 commands. These were exactly the commands which the Unix command `file` listed as "Mach-O 64-bit executable x86_64".

In addition, Apple found three other such binaries in `tlpkg/installer`: `lz4`, `wget`, `xz`.

I used the codesign script on these 33 binaries and submitted BasicTEX again to Apple for notarization. Approved!

6.3 Case 2: Ghostscript

Ghostscript only has two binaries, `gs-X11` with X11 support and `gs-noX11` without X. We install a symbolic link named `gs` to the appropriate binary.

I ran `codesign` on `gs-X11` and `gs-noX11` and submitted to Apple. Apple notarized the install package. But when the package was used to install Ghostscript, `gs` refused to run. Why?

Originally, Apple supplied an optional install package for X11. But their package was often out of date, so a mutual decision was made for a third party to supply X11 for the Macintosh as open source. Consequently, `gs-X11` links in a third party library, which is not allowed for hardened runtimes. Resigning `gs-X11` and claiming an entitlement for such linking solved the problem.

6.4 Case 3: `biber`

The `biber` binary is so complicated that TEX Live builders do not compile it. Instead the author submits binaries. The `codesign` script didn't work with this binary. I contacted the author, Philip Kime. A month later he sent a binary which worked. I suspect Kime knows a lot more about notarization than I do now.

6.5 Case 4: The big enchilada

Finally it was time to notarize the full TEX Live. I hardened `xz`, `wget`, `lz4`, and all the binaries in `bin/x86_64-darwin` which were not links and reported to be "Mach-0 64-bit executables" by `file`. Tests revealed that two of these binaries needed an exception for X11: `mf` and `xdvi-xaw`. I submitted the package to Apple. It was rejected.

A big difference between BasicTEX and the full TEX Live is that the second package has documentation provided by package makers. This documentation comes in a wide variety of formats: source files for illustrations, zip files, and so forth. When Apple tests an install package for viruses, does it unzip files and look inside? Yes, it does. Does it examine illustration source files? Yes, it does that too. So lots of things could go wrong.

Luckily, Apple provided clear explanations for rejection, and it turned out that MacTEX had only three problems:

- In `texmf-dist/doc/support/ctan-o-mat`, one file is given an extension `.pkg`. Apple believes that a file with extension `.pkg` is an install package, and this package was not signed. It turned out to be an ordinary text file.
- In `texmf-dist/doc/latex/codepage`, Apple could not unzip the file `demo.zip`.
- In `texmf-dist/source/latex/stellenbosch`, there is a zip file named `USlogos-4.0-src.zip` containing two CorelDraw source files for illustrations. Apple did not recognize these source files and flagged them.

The three problems were easy to work around. Bug reports were also sent to Apple so they can improve the notarization machinery.

7 Status of notarization for MacTEX-2019

Fully notarized install packages for MacTEX-2019, BasicTEX-2019, and Ghostscript-9.27 are available on the web for testing. Indeed, the Ghostscript-9.27 package on CTAN is already notarized. The MacTEX-2019 and BasicTEX-2019 packages will be moved to CTAN, replacing the original packages, in late summer just before Catalina is released.

TEX Live Utility, LATEXiT, and BibDesk are not in the notarized MacTEX-2019 because they are applications rather than command line programs, *so their authors must sign and notarize them.* This has not yet happened. If these authors used the XCode which comes with Mojave, these steps would be trivial, but they use an older XCode. We are working with the authors but have nothing to report.

8 Technical details

I end with some technical details for others who may need to deal with these issues on the Macintosh. I'll explain how to sign install packages and how to notarize such packages. Then I'll list the six runtime entitlements and seven resource access entitlements from an official Apple document.

8.1 Signing an install package

Signing requires developer status from Apple, which costs $100 a year. Certificate information and security codes are kept on Apple's KeyChain, and automatically retrieved by the signing software when needed. If you buy a new machine or install a new system, you must transfer this information to the new system. XCode makes this easy *if* you know what mysterious icon to click.

Signing applications happens automatically in XCode as part of the build process. Signing install packages is done on the command line. The command here signs `Temp.pkg` and writes the signed package `Basic.pkg`.

```
productsign  \
 --sign "Developer ID Installer: Richard Koch" \
 Temp.pkg Basic.pkg
```

8.2 Notarizing an install package

Notarization of install packages is done on the command line, and is somewhat trickier. Below are the crucial commands. The first command sends an install package to Apple to be notarized. If uploading succeeds, this command returns an identifier which I symbolize with `YYYY`; it is actually much longer.

```
xcrun altool --notarize-app \
  --primary-bundle-id \
       "org.tug.mactex.basictex" \
 --username "koch@uoregon.edu" \
 --password "XXXX" \
 --file BasicTeX.pkg
```

When Apple is finished, it sends a brief email stating whether notarization was successful. If there were errors, this second command asks for a detailed list of errors. The command returns a url, and the error list will then appear in a browser pointed to this url.

```
xcrun altool --notarization-info YYYY \
  --username  "koch@uoregon.edu" \
  --password "XXXX"
```

If notarization was successful, this third command staples the certificate to the install package, producing a notarized package:

```
xcrun stapler staple "BasicTeX.pkg"
```

In these commands, `altool` is a command line tool which communicates with Apple. This communication is normally protected using two-factor authentication, but that is not convenient for command line work. So before using `altool`, Apple asks developers to log into their account and give `altool` a temporary password. The symbol `XXXX` in the first and second commands represents this password.

The value `org.tug.mactex.basictex` in the first command identifies the install package for the notification process, but need not correspond to any similar string in the package. So the identifier can be selected randomly.

8.3 Runtime entitlements

All entitlements are boolean values; all keys start with `com.apple.security`, not shown here for brevity.

Allow Execution of JIT-compiled Code: whether the app may create writable and executable memory using the `MAP_JIT` flag. Key: `.cs.allow-jit`

Allow Unsigned Executable Memory: whether the app may create writable and executable memory without using the `MAP_JIT` flag. Key: `.cs.allow-unsigned-executable-memory`

Allow DYLD Environment Variables: whether the app may be impacted by DYLD environment variables, which can be used to inject code into the process. Key: `.cs.allow-dyld-environment-variables`

Disable Library Validation: whether the app may load plug-ins or frameworks signed by other developers. Key: `.cs.disable-library-validation`

Disable Executable Memory Protection: whether to disable code signing protections while launching the app. Key: `.cs.disable-executable-page-protection`

Debugging Tool: whether the app is a debugger and may attach to other processes or get task ports. Key: `.cs.debugger`

8.4 Resource access entitlements

Audio Input: whether the app may record audio using the built-in microphone and access audio input using Core Audio. Key: `.device.audio-input`

Camera: whether the app may capture movies and still images using the built-in camera. Key: `.device.camera`

Location: whether the app may access location information from Location Services. Key: `.personal-information.location`

Address Book: whether the app may have read-write access to contacts in the user's address book. Key: `.personal-information.addressbook`

Calendars: whether the app may have read-write access to the user's calendar. Key: `.personal-information.calendars`

Photos Library: whether the app may have read-write access to the user's Photos library. Key: `.personal-information.photos-library`

Apple Events: whether the app may send Apple Events to other apps. Key: `.automation.apple-events`

◇ Richard Koch
koch (at) math dot uoregon dot edu
http://math.uoregon.edu/koch/