Debugging HTTP Proxies and Certificate Transparency

This thread has been locked by a moderator.

182

I regularly see folks confused by this issue, both here on DevForums and in DTS incidents, so I thought I'd write it up publicly.

If you have questions or comments, start new thread here on DevForums. Tag it with *Foundation*, *CFNetwork*, and *Security* so that I see it.

Share and Enjoy
—
Quinn "The Eskimo!" @ Developer Technical Support @ Apple
`let myEmail = "eskimo" + "1" + "@" + "apple.com"`

Debugging HTTP Proxies and Certificate Transparency

Some developers think that Certificate Transparency (CT) will prevent the user from using a debugging HTTP proxy to inspect the requests issued by their app. That's not the case. This post explains why, and offers advice as to what you might do about that.

About debugging HTTP proxies

A debugging HTTP proxy works using a process known as **TLS inspection** [1]. In rough terms:

1. The user sets up their debugging HTTP proxy.
2. The proxy creates its own, internal certificate authority (CA).
3. The user installs that CA's root certificate on their Apple device.
4. And configures their device to use their proxy.
5. When the device makes an outgoing HTTPS request, it makes a TLS connection through the proxy.
6. A normal proxy would pass the connection through to the origin server. A debugging proxy does not. Rather, it uses the CA it set up in step 2 to create its own certificate for the origin server. It has the private key for that certificate, and thus can recover the plaintext from the TLS connection.
7. The proxy then makes its own TLS connection to the origin server and forwards the plaintext down that.

Debugging HTTP proxies are a great tool for app developers. For references to some of the more popular ones, see Taking Advantage of Third-Party Network Debugging Tools.

Note Historically such proxies were a critical implement in any app developer's debugging toolkit. These days, however, it's often easier to use Instruments to inspect HTTP requests. See Analyzing HTTP Traffic with Instruments.

[1] Well, different folks use different names, but *TLS inspection* is the name that I prefer. And it seems that I'm good company (-:

https://www.schneier.com/blog/archives/2019/11/the_nsa_warns_o.html

That post's link to the original NSA article is broken, but you can view it on the Wayback Machine:

https://web.archive.org/web/20191119195359/https://media.defense.gov/2019/Nov/18/2002212783/-1/-1/0/MANAGING%20RISK%20FROM%20TLS%20INSPECTION_20191106.PDF

About Certificate Transparency

Certificate Transparency is a technology designed to protect the user from misissued certificates. Apple platforms support CT out of the box. For more on that, see the links in Networking Resources.

Historically CT was not enabled by default. However, an app using `URLSession` could opt in to it by adding the `NSRequiresCertificateTransparency` property to their App Transport Security (ATS) configuration.

This is no longer necessary. Starting with iOS 16 and its aligned releases, CT is always on, and thus the `NSRequiresCertificateTransparency` property is no longer useful.

Crossing the streams

Some developers think that Certificate Transparency will prevent the user from using a debugging HTTP proxy to inspect the requests issued by their app. That's not the case. Apple's CT support is focused on protecting the user from certificates misissued by the built-in trusted CAs [1]. It's not relevant for certificates issued by a custom CA, and thus it doesn't block the operation of a debugging HTTP proxy.

If you want block the operation of a debugging HTTP proxy, the easiest path forward is the `NSPinnedDomains` property. For the details, see its documentation.

IMPORTANT Some enterprise environments rely on TLS inspection, and it's likely that this will prevent your app from working in such environments.

If your app supports older systems, or the rules of your pinning policy can't be expressed by the `NSPinnedDomains` property, you can implement your own policy by overriding HTTPS server trust evaluation. For advice on how to do this with `URLSession`, see Performing Manual Server Trust Authentication.

[1] That is, a CA whose root certificate is trusted by the system by default. For a link to the Apple Support article that lists the built-in trusted root certificates, see Networking Resources.

Foundation Security CFNetwork

Reply

Posted 2 months ago by eskimo

Add a Comment