Customising TLS Server Trust Evaluation Considered Harmful

A common question here on DevForums runs something along the lines of “How do I talk to a server with a self-signed certificate?” The answer is “It’s possible, but it’s best to avoid self-signed certificates if you can.” This post explains that last point in detail.

In my experience there are four common reasons for using a self-signed certificate:

- Cost or inconvenience of a CA-issued certificate
- Enterprise
- Testing
- Low infrastructure (for example, talking to an accessory on the local network that has no fixed DNS name)

All of these scenarios except the last one have alternatives that are clearly better. I’ll explain each in turn below. Before doing that, however, I want to outline why self-signed certificates are such a problem.

Note If you’re not familiar with the basics of TLS, I recommend that you start by reading my TLS for App Developers post first.

Problems

Talking to a server with a self-signed certificate is tricky because the OS does not trust such a server by default. Thus, to successfully connect you have to **customise TLS server trust evaluation**. This presents some significant challenges:

- It is not possible to do in all cases; some subsystems provide no hooks for customising their TLS server trust evaluation
- You typically end up needing a bunch of extra code
- You need to design that code to maintain the security properties of TLS, and it’s hard to create such a design without a deep understanding of how TLS server trust evaluation works
- Any bug in the design or implementation of your code could expose you to serious security vulnerabilities

WARNING Do not discount the risk here. A number of developers, including some household names, have had to publish an emergency update to their app because they customised TLS server trust evaluation incorrectly. In some cases this problem was not found for years, exposing their customers’ data for all that time.

The best way to avoid these issues is to *not customise TLS server trust evaluation*. The next section outlines the common reasons for doing this and, in most cases, points to a better approach.

Common Scenarios and Their Solutions

This section describes the four common scenarios where folks think they need to work with self-signed certificates, and in most cases offers a better way forward.

Cost or Inconvenience of a CA-Issued Certificate

Some developers are reluctant to give there server a CA-issued certificate because of the cost or inconvenience that involves. While that may have been true in the past, these days it’s relatively cheap and easy to get a CA-issued certificate for your server. The obvious way forward here is Let’s Encrypt, which is both free and convenient, but there are lots of other low-cost options.

Enterprise

Developers working within an enterprise will often try to use a self-signed certificate for their server because their server is not on the public Internet, which makes it hard to get a CA-issued certificate for it.

The solution here is to run a CA within your enterprise and have that CA issue your server a certificate. You can then push the CA’s root certificate to your devices via MDM. This tells the device to trust certificates issued by that CA, including the certificate used by your server.

Indeed, many enterprises already run a CA for other reasons — for example, to issue certificates allowing users to sign their email — and you can piggyback on that infrastructure for your server.

Testing

Some folks think that you need to use a self-signed certificate for testing. Fortunately that’s not the case. Rather, you can set up your own test CA, have it issue a certificate for your server, and install your test CA’s root certificate on your device. If you don’t have existing experience with running a CA, there are facilities built in to the Mac to do this; see Technote 2326 Creating Certificates for TLS Testing.

WARNING Overriding TLS server trust evaluation for testing is particularly insidious because it’s very easy to leave that code enabled in your production build, thus exposing your final product to serious security vulnerabilities.

Low Infrastructure

TLS assumes that the server will exist on the wider network at some fixed DNS name or IP address [1]. This presents some serious challenges when working in a low-infrastructure environment, for example:

- Implementing a peer-to-peer networking app
- Talking to a server embedded within an accessory on the local network

In such situations you can’t avoid the requirement to customise TLS server trust evaluation.

If you find yourself in this situation then you must carefully design your customisation to maintain as many of TLS’s security properties as possible.

WARNING Disabling TLS server trust evaluation completely is *never* a good idea.

At a minimum I recommend that you implement the ‘initially confirm’ approach. You may be familiar with this approach from SSH, where the SSH client asks you to confirm the identity of the server when you first connect and then, on subsequent connects, it checks the server’s identity and sets off warning bells if it’s changed.

However, there may be more secure approaches that work for your specific situation. If you’d like to discuss such approaches, please start a new thread here on Core OS > Networking, making sure to include a description of your requirements.

[1] Technically this isn’t an attribute of TLS, but of the public key infrastructure (PKI) that TLS depends on.

Customising TLS Server Trust Evaluation

If, after reading through all of the above, you decide that customising TLS server trust evaluation is necessary, this section describes what you need to do.

On modern systems there are two aspects to TLS server trust evaluation:

- All parts of the OS implement default (RFC 2818) TLS server trust evaluation
- High-level APIs, including

  ```
  NSURLSession
  ```

  , are subject to the additional checks imposed by App Transport Security (ATS)

To successfully customise TLS server trust evaluation you must deal with both of these aspects:

- For the default TLS server trust evaluation, read Technote 2232 HTTPS Server Trust Evaluation.
- If you’re using a high-level API, and thus have to deal with ATS, you should read my App Transport Security pinned post, which has a bunch of info about ATS and includes references to the official ATS documentation.

Share and Enjoy — Quinn “The Eskimo!” Apple Developer Relations, Developer Technical Support, Core OS/Hardware

```
let myEmail = "eskimo" + "1" + "@apple.com"
```

Network

Asked 2 years ago by eskimo

Reply to this question