The If Works

by James Coglan

Open source
Books
Conference talks
Podcast interviews

Missing TLS certificate verification in Faye

The `Faye::WebSocket::Client` class uses the `EM::Connection#start_tls` method in EventMachine to implement the TLS handshake whenever a `wss:` URL is used for the connection. This method does not implement certificate verification by default, meaning that it does not check that the server presents a valid and trusted TLS certificate for the expected hostname. That means that any `wss:` connection made using this library is vulnerable to a man-in-the-middle attack, since it does not confirm the identity of the server it is connected to.

This has been a requested feature in EventMachine for many years now; see for example #275, #378, and #814. In June 2020, em-http-request published an advisory related to this problem and fixed it by implementing TLS verification in their own codebase; although EventMachine does not implement certificate verification itself, it provides an extension point for the caller to implement it, called `ssl_verify_peer`. Based on this implementation, we have incorporated similar functionality into faye-websocket for Ruby, such that we use the `OpenSSL` module to perform two checks:

- The chain of certificates presented by the server is valid and ultimately trusted by your root certificate set – either your system default root certificates, or a set provided at runtime
- The final certificate presented by the server is valid for the hostname used in the request URI; if the connection is made via a proxy we use the hostname from the request, not the proxy's hostname

After implementing verification in v1.1.6, em-http-request has elected to leave the `:verify_peer` option switched off by default. We have decided to *enable* this option by default in faye-websocket, but are publishing a minor release with added functionality for configuring it. We are mindful of the fact that this may break existing programs, but we consider it much more important that all clients have TLS verification turned on by default. A client that is not carrying out verification is either:

- talking to the expected server, and will not break under this change
- being attacked, and would benefit from being alerted to this fact
- deliberately talking to a server that would be rejected by verification

The latter case includes situations like talking to a non-public server using a self-signed certificate. We consider this use case to be "working by accident", rather than functionality that was actively supported, and it should be properly and explicitly supported instead. To that end, we have added two new options to the `Faye::WebSocket::Client` constructor: `tls.root_cert_file`, and `tls.verify_peer`.

The `:root_cert_file` option lets you provide a different set of root certificates in situations where you don't want to use your system's default root certificates to verify the remote host. It should be a path or an array of paths identifying the certificates to use instead of the defaults.

```
client = Faye::WebSocket::Client.new('wss://example.com/', [], tls: {
  root_cert_file: 'path/to/certificate.pem'
})
```

The `:verify_peer` option lets you turn verification off entirely. This should be a last resort and we recommend using the `:root_cert_file` option if possible.

```
client = Faye::WebSocket::Client.new('wss://example.com/', [], tls: {
  verify_peer: false
})
```

To get the new behaviour, please upgrade to v0.11.0 of the Rubygems package. There are, unfortunately, no workarounds for this issue, as you cannot enable `:verify_peer` in EventMachine unless the calling library contains an implementation of `ssl_verify_peer` that actually checks the server's certificates.

The messaging product Faye uses em-http-request and faye-websocket in the Ruby version of its client. The first request a client makes is always sent via normal HTTP, but later messages may be sent via WebSocket. Therefore it is vulnerable to the same problem that these underlying libraries are, and we need both libraries to support TLS verification before Faye can claim to do the same. Your client would still be insecure if its initial HTTPS request was verified, but later WebSocket connections were not.

Now that both libraries are fixed, we are releasing Faye v1.4.0, which enables verification by default and provides a way to opt out of it:

```
client = Faye::Client.new('https://example.com/', tls: { verify_peer: false })
```

Unfortunately we can't offer an equivalent of the `:root_cert_file` option, because this is only supported by faye-websocket, not em-http-request. If you need to talk to servers whose certificates are not recognised by your default root certificates, then you need to add its certificate (or another one that can verify it) to your system's root set.

The same functionality is now supported in the Node.js version, with a `tls` option whose values will be passed to the `https` and `tls` modules as appropriate when making connections. For example, you can provide your own CA certificate:

```
var client = new faye.Client('https://example.com/', {
  tls: {
    ca: fs.readFileSync('path/to/certificate.pem')
  }
});
```

For further background information on this issue, please see faye#524 and faye-websocket#129. We would like to thank Tero Marttila and Daniel Morsing for providing invaluable assistance and feedback on this issue.

Posted on July 31, 2020. This entry was posted in Faye, Ruby. Bookmark the permalink.

← ReDoS vulnerability in websocket-extensions

→ Reading and writing, part 1: locations and locks