The ISP Column
*A monthly column on things Internet*

April 2020
Geoff Huston

# The Wrong Certificate:
## Apache, Let's Encrypt and OpenSSL

I'm constantly impressed by the rather complex intricacies that are associated with running your own web server these days. A recent source of these complexities has been the PKI, the security infrastructure used to maintain secure connections over the network, and I'd like to recount my experience here, in case any others encounter the same seemingly inexplicable behaviours in their secure web service configurations.

For me, all this started with the thought that after some years of trying to ignore the topic, it was time to upgrade my web server system to include a rewrite rule to push all HTTP (unencrypted) requests to use secured sessions using Transport Layer Security (TLS). It wasn't just time to do this, it was well beyond time! At the same time, I thought it would be a good idea to upgrade the underlying web server hardware to update the web platform to meet current traffic demands. Accordingly, I started down the path of migration of the entire web service environment to a different hardware platform running up to date versions of the software set. All thoroughly laudable objectives, so I thought!

That decision meant that many aspects of the service had to change. The operating system on the previous platform was stuck on FreeBSD 11.2, and the current release version is 12.1. My hosting framework uses virtual hosting on an Apache 2.4 configuration. The old host used Apache version 2.4.35, while the new system uses Apache version 2.4.41. On the old host all the virtual hosts were loaded into a single Apache instance listening on all interfaces. This time around the virtual hosts were divided across two Apache instances, each listening on different IP addresses in order to perform some level of load isolation in the platform.

There were a number of moving parts in this transition, and we were careful to check the integrity of the new service as we performed each step of the migration.

## Let's Encrypt Certificates

One of the issues here was ensuring that the SSL configuration had not been broken. As well as splitting up the virtual host definitions the certificate declarations in Apache had also been changed in this move. The old system used a configuration:

```
SSLCertificateFile "/dir/cert.pem"
SSLCertificateChainFile "/dir/fullchain.pem"
```

It appears that these days we should be using a slightly different template:

```
SSLCertificateFile "/dir/cert.pem"
SSLCertificateChainFile "/dir/chain.pem"
```

The difference is that the Let's Encrypt "fullchain" certificate has both the certificate issued by Digital Signature Trust Co., that certifies Let's Encrypt and the certificate issued by Let's Encrypt that certifies my domain, while the "chain" certificate only has the parent certificate issued by Digital Signature Trust.

In theory this should not matter, but we decided to use the updated template anyway. We installed the software, ran up the test (the `-resolve` option in `curl` is very useful here to test the setup prior to actually switching over the entire environment), then changed the DNS and started further tests on the new web platform.

Accessing the relocated service with various browsers presented a few problems with using the correct syntax in the changed configuration files, but once these were sorted out it all looked good.

Except for one test.

## OpenSSL

The test we were using was a client connection using `OpenSSL`. The command was:

```
$ openssl s_client -connect x.labs.apnic.net:443
```

The output is voluminous, but the part of interest here is the certificate chain:

```
$ openssl s_client -connect x.labs.apnic.net:443
CONNECTED(00000003)
depth=2 O = Digital Signature Trust Co., CN = DST Root CA X3
verify return:1
depth=1 C = US, O = Let's Encrypt, CN = Let's Encrypt Authority X3
verify return:1
depth=0 CN = y.labs.apnic.net
verify return:1
---
Certificate chain
 0 s:/CN=y.labs.apnic.net
   i:/C=US/O=Let's Encrypt/CN=Let's Encrypt Authority X3
 1 s:/CN=y.labs.apnic.net
   i:/C=US/O=Let's Encrypt/CN=Let's Encrypt Authority X3
 2 s:/C=US/O=Let's Encrypt/CN=Let's Encrypt Authority X3
   i:/O=Digital Signature Trust Co./CN=DST Root CA X3
---
```

There is something odd here. We are trying to connect to the virtual host at `x.labs.apnic.net` over port 443, yet the certificate being offered to set up the connection is something entirely different. It's the certificate for the hostname `y.labs.apnic.net`. It's not a completely random choice, as there is also a virtual host `y.labs.apnic.net` defined in the Apache Virtual Host configuration

Why are we seeing the Apache server offer a certificate for `y.labs.apnic.net` when we asked the `OpenSSL` client to connect to `x.labs.apnic.net`?

The first step in trying to understand this situation was to try the same `openssl s_client` command on different hosts, and while some other hosts replicated the same anomalous response shown above, other hosts behaved as we'd expected:

```
$ openssl s_client -connect x.labs.apnic.net:443
CONNECTED(00000003)
depth=2 O = Digital Signature Trust Co., CN = DST Root CA X3
verify return:1
depth=1 C = US, O = Let's Encrypt, CN = Let's Encrypt Authority X3
```

```
verify return:1
depth=0 CN = x.labs.apnic.net
verify return:1
---
Certificate chain
 0 s:CN = x.labs.apnic.net
   i:C = US, O = Let's Encrypt, CN = Let's Encrypt Authority X3
 1 s:CN = x.labs.apnic.net
   i:C = US, O = Let's Encrypt, CN = Let's Encrypt Authority X3
 2 s:C = US, O = Let's Encrypt, CN = Let's Encrypt Authority X3
    i:O = Digital Signature Trust Co., CN = DST Root CA X3
---
```

Here certificates 0 and 1 in the certificate chain are what we expected: the subject name is `x.labs.apnic.net`. It was clear that this was consistent behaviour. Each host we used showed one of other of these behaviours, but each individual test host did not switch between them. It either used the expected certificate or it used a different one to set up the SSL session, and it did so repeatedly.

There are two questions at this point in time:

- Why is the `y.labs.apnic.net` certificate being presented?
- Why isn't the `x.labs.apnic.net` certificate being presented?

## Why is the y.labs.apnic.net certificate being presented?

The first question can be answered by looking carefully through the Apache documentation. The best explanation can be found on the Apache Wiki on the section on name-based virtual hosts (https://cwiki.apache.org/confluence/display/HTTPD/NameBasedSSLVHostsWithSNI)

Virtual hosts in HTTP are supported using the HTTP `Host:` parameter in the initial HTTP request. Here's a dump of such a connection:

```
$ curl -Lv http://x.labs.apnic.net
* Connected to x.labs.apnic.net (2401:2000:6660::109) port 80 (#0)
> GET / HTTP/1.1
> Host: x.labs.apnic.net
> User-Agent: curl/7.64.1
> Accept: */*
>
< HTTP/1.1 200 OK
    ...
```

The initial TCP connection with the port 80 Apache server is common to all virtual hosts, and it's the `Host:` part of the GET directive that establishes the intended virtual host parameter of the session.

This approach will not work for SSL connections over port 443. The secure session must be established *before* the HTTP connection can be set up, so the Apache server needs to the told which certificate to use before the client can issue the `Host` directive inside the HTTP session.

This is achieved in TLS through the use of the Server Name Indication (SNI) field in the initial TLS handshake (RFC 4366). The client tells the server in the initial unencrypted exchange the name of the server it wants to connect to so that the server can select the 'right' certificate to use to respond.

> Yes, the unencrypted SNI field in the initial TLS exchange is an information leak and there are moves to encrypt this field in TLS 1.3, but that is a different story for a different time.

What if there is no SNI field in this `ClientHello` part of the TLS handshake?

If you have turned on Apache's directive `SSLStrictSNIVHostCheck` then no SNI implies no connection will be made. But I didn't even know that such a directive existed, and I certainly had not set it to "on". I was using the default action, and in Apache's case the lack of an SNI field in the TLS handshake causes Apache to use the first Virtual Host in the configuration set to respond. In my configuration that was `y.labs.apnic.net`. When I changed the configuration to use a different initial Virtual Host, then the `openssl s_client` command prompted the server to pass back this different certificate.

That's enough to conclude that we have a plausible cause for this behaviour. The `y.labs.apnic.net` certificate is being used because the `openssl s_client` command did not include a SNI value. This triggered my Apache server to use the first Virtual Host context in its local configuration to complete the TLS handshake, as I had not enabled strict strict SNI checking so Apache's default action was used. This happened to be the entry for `y.labs.apnic.net`.

We now have an answer to the first question. This is the first Virtual Host block in the Apache SSL configuration section, and Apache is using the default behaviour to select this first Virtual Host context when no SNI value is given in the `ClientHello` part of the TLS setup.

## Why isn't the x.labs.apnic.net certificate being presented?

Recent client browsers always offer SNI values, as does `curl`, so when I use other diagnostic tools such as `curl` then everything looks ok.

```
$ curl -Lv https://x.labs.apnic.net/
*   Trying 2401:2000:6660::109...
* TCP_NODELAY set
*   Trying 203.133.248.109...
* TCP_NODELAY set
* Connected to x.labs.apnic.net (2401:2000:6660::109) port 443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* Cipher selection:
ALL:!EXPORT:!EXPORT40:!EXPORT56:!aNULL:!LOW:!RC4:@STRENGTH
* successfully set certificate verify locations:
*   CAfile: /etc/ssl/certs/ca-certificates.crt
  CApath: /etc/ssl/certs
* TLSv1.2 (OUT), TLS header, Certificate Status (22):
* TLSv1.2 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS change cipher, Client hello (1):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS change cipher, Client hello (1):
* TLSv1.2 (IN), TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / ECDHE-RSA-AES256-GCM-SHA384
* ALPN, server accepted to use http/1.1
* Server certificate:
*  subject: CN=x.labs.apnic.net
*  start date: Apr  3 21:03:30 2020 GMT
*  expire date: Jul  2 21:03:30 2020 GMT
*  subjectAltName: host "x.labs.apnic.net" matched cert's
"x.labs.apnic.net"
*  issuer: C=US; O=Let's Encrypt; CN=Let's Encrypt Authority X3
*  SSL certificate verify ok.
> GET / HTTP/1.1
> Host: x.labs.apnic.net
> User-Agent: curl/7.52.1
> Accept: */*
>
< HTTP/1.1 200 OK
```

This connection was established on the platform that had an issue with the `openssl s_client` command. As the transcript shows, the connection opens with TLS version 1.2.

This implies that the problem appears to be related to the `openssl s_client` command.

What happens when we use a host where the `openssl s_client` command worked as expected?

```
$ curl -Lv https://x.labs.apnic.net/
* TCP_NODELAY set
* Expire in 149999 ms for 3 (transfer 0x555f219aaf50)
* Connected to x.labs.apnic.net (2401:2000:6660::109) port 443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* successfully set certificate verify locations:
*   CAfile: none
  CApath: /etc/ssl/certs
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.3 (IN), TLS handshake, Encrypted Extensions (8):
* TLSv1.3 (IN), TLS handshake, Certificate (11):
* TLSv1.3 (IN), TLS handshake, CERT verify (15):
* TLSv1.3 (IN), TLS handshake, Finished (20):
* TLSv1.3 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.3 (OUT), TLS handshake, Finished (20):
* SSL connection using TLSv1.3 / TLS_AES_256_GCM_SHA384
* ALPN, server accepted to use http/1.1
* Server certificate:
*  subject: CN=x.labs.apnic.net
*  start date: Apr  3 21:03:30 2020 GMT
*  expire date: Jul  2 21:03:30 2020 GMT
*  subjectAltName: host "x.labs.apnic.net" matched cert's
"x.labs.apnic.net"
*  issuer: C=US; O=Let's Encrypt; CN=Let's Encrypt Authority X3
*  SSL certificate verify ok.
> GET / HTTP/1.1
> Host: x.labs.apnic.net
> User-Agent: curl/7.64.0
> Accept: */*
>
* TLSv1.3 (IN), TLS handshake, Newsession Ticket (4):
* TLSv1.3 (IN), TLS handshake, Newsession Ticket (4):
* old SSL session ID is stale, removing
< HTTP/1.1 200 OK
```

It's almost the same, but now the session is using TLS version 1.3. It looks like there is some issue with versions `of OpenSSL` libraries on these hosts.

Failing hosts:

```
$ openssl version
OpenSSL 1.1.0l  10 Sep 2019
```

Working hosts:

```
$ openssl version
OpenSSL 1.1.1d  10 Sep 2019
```

It seems that something changed in `OpenSSL` between versions 1.1.0 and 1.1.1. Let's look at the documentation for OpenSSL 1.1.1. (Well just the `s_client` documentation, as OpenSSL is the Texas Chainsaw Massacre of crypto functions and the library has a massive set of commands and options!)

> -servername name
>
> Set the TLS SNI (Server Name Indication) extension in the ClientHello message to the given value. If -servername is not provided, the TLS SNI extension will be populated with the name given to -connect if it follows a DNS name format. If -connect is not provided either, the SNI is set to "localhost". This is the default since OpenSSL 1.1.1.
>
> Even though SNI should normally be a DNS name and not an IP address, if -servername is provided then that name will be sent, regardless of whether it is a DNS name or not.
>
> https://www.openssl.org/docs/man1.1.1/man1/openssl-s_client.html

Hmm. If this was the default since OpenSSL 1.1.1 what was the default before that version? Here's the same manual entry for version 1.1.0 of OpenSSL `s_client`.

> -servername name
>
> Set the TLS SNI (Server Name Indication) extension in the ClientHello message.
>
> https://www.openssl.org/docs/man1.1.0/man1/openssl-s_client.html

The implication is now clear: If the TLS SNI is not set using this directive, then there is no SNI extension in the `ClientHello` message in this version of OpenSSL.

It looks like we have found the problem. Let's check on an OpenSSL 1.1.0 client, adding a value for the servername parameter:

```
$ openssl s_client -connect x.labs.apnic.net:443 -servername
x.labs.apnic.net
CONNECTED(00000003)
depth=2 O = Digital Signature Trust Co., CN = DST Root CA X3
verify return:1
depth=1 C = US, O = Let's Encrypt, CN = Let's Encrypt Authority X3
verify return:1
depth=0 CN = x.labs.apnic.net
verify return:1
---
Certificate chain
 0 s:/CN=x.labs.apnic.net
   i:/C=US/O=Let's Encrypt/CN=Let's Encrypt Authority X3
 1 s:/CN=x.labs.apnic.net
   i:/C=US/O=Let's Encrypt/CN=Let's Encrypt Authority X3
 2 s:/C=US/O=Let's Encrypt/CN=Let's Encrypt Authority X3
   i:/O=Digital Signature Trust Co./CN=DST Root CA X3
```

Let's confirm that by giving a `servername` parameter value that is not the same as the connect name parameter:

```
$ openssl s_client -connect x.labs.apnic.net:443 -servername
dmm.labs.apnic.net
CONNECTED(00000003)
depth=2 O = Digital Signature Trust Co., CN = DST Root CA X3
verify return:1
depth=1 C = US, O = Let's Encrypt, CN = Let's Encrypt Authority X3
verify return:1
depth=0 CN = dmm.labs.apnic.net
verify return:1
---
```

```
Certificate chain
 0 s:/CN=dmm.labs.apnic.net
   i:/C=US/O=Let's Encrypt/CN=Let's Encrypt Authority X3
 1 s:/C=US/O=Let's Encrypt/CN=Let's Encrypt Authority X3
   i:/O=Digital Signature Trust Co./CN=DST Root CA X3
---
```

We now have an answer to the second question. It's the way we were using `openssl -s_client` on hosts with `OpenSSL` version 1.1.0. By omitting the `-servername` argument we triggered this behaviour.

## The Right Certificate

We've found the issue. There was nothing wrong with Apache, nothing wrong with the Let's Encrypt certificates, nothing wrong with browsers. There wasn't even anything that was "wrong" in OpenSSL.

There was however a subtle inconsistency in the default behaviour of one of the diagnostic tools in the OpenSSL library across versions.

Security is meant to help us build more robust systems. But with the PKI we have an incredible number of moving parts and now even within each component different versions of diagnostic tools may have different default behaviours.

We often hear that "security is hard". But my takeaway from a day of pushing and prodding at web server configurations and wading through search results is slightly different. These days "security is obtuse" and that's something that just should not be happening!

## Disclaimer

The above views do not necessarily represent the views or positions of the Asia Pacific Network Information Centre.

## Author

Geoff Huston AM, B.Sc., M.Sc., is the Chief Scientist at APNIC, the Regional Internet Registry serving the Asia Pacific region.

*www.potaroo.net*