Testing a TLS connection

Fabio Massimo Ercoli *

December 25th 2024

1 TLS handshake runs

I created the script file *tls-checker.sh* to test the TLS handshakes on 20 different sites.

`google.com, facebook.com, redhat.com, infinispan.org, almaviva.it, www.eng.it, edenred.it, esa.int, theguardian.com, repubblica.it, huffingtonpost.it, quotidianolavoce.it, baraondanews.it, mit.edu, stanford.edu, uniroma1.it, unicusano.it, chiaveorgonica.it, diocesiportosantarufina.it, hwpviewer.hbedu.co.kr`

The core of the script is the *checkTLS* function:

```
#!/usr/bin/env bash
set -e
openssl version

function checkTLS() {
        start=`date +%s%N`
        echo | openssl s_client -connect $1:443 2>/dev/null
        end=`date +%s%N`
        runtime=$((end-start))
        echo "$1 => $runtime"
}

checkTLS google.com
checkTLS facebook.com
...
```

I collected the output in a text file *tls-checkers.txt*

```
./tls-checker.sh > tls-checkers.txt
```

*More on me: my GitHub page - my Linkedin page.

2 TLS handshake times

From the previous result it is possible to extract the times, they are collected as microseconds (1 microsecond = 10^{-9} seconds).

```
cat tls-checkers.txt | grep "=>"
cat tls-checkers.txt | grep "New, "
```

Here is my result:

Connection	Time	TLSv	Cipher
google.com	115.264.962	TLSv1.3	TLS AES 256 GCM SHA384
facebook.com	066.053.836	TLSv1.3	TLS CHACHA20 POLY1305 SHA256
redhat.com	568.495.098	TLSv1.3	TLS AES 256 GCM SHA384
infinispan.org	097.765.850	TLSv1.3	TLS AES 128 GCM SHA256
almaviva.it	216.864.263	TLSv1.2	ECDHE-RSA-AES128-GCM SHA256
www.eng.it	109.731.688	TLSv1.3	TLS AES 128 GCM SHA256
edenred.it	108.295.706	TLSv1.3	TLS AES 256 GCM SHA384
esa.int	204.531.698	TLSv1.2	ECDHE-RSA-AES256-GCM SHA384
theguardian.com	123.894.304	TLSv1.3	TLS AES 128 GCM SHA256
repubblica.it	86.482.951	TLSv1.3	TLS AES 128 GCM SHA256
huffingtonpost.it	269.442.781	TLSv1.2	ECDHE-ECDSA-AES128-GCM SHA256
quotidianolavoce.it	661.609.322	TLSv1.3	TLS AES 256 GCM SHA384
baraondanews.it	170.123.645	TLSv1.3	TLS AES 256 GCM SHA384
mit.edu	1040.751.680	TLSv1.2	ECDHE-RSA-AES256-GCM SHA384
stanford.edu	912.750.222	TLSv1.2	AES128-GCM SHA256
uniroma1.it	137.912.714	TLSv1.2	ECDHE-RSA-AES256-GCM SHA384
unicusano.it	081.213.441	TLSv1.3	TLS AES 128 GCM SHA256
chiaveorgonica.it	162.517.465	TLSv1.3	TLS AES 256 GCM SHA384
diocesiportosantarufina.it	141.360.897	TLSv1.3	TLS AES 256 GCM SHA384
hwpviewer.hbedu.co.kr	1636.988.593	TLSv1.3	TLS AES 256 GCM SHA384

3 Describes two connections in detail

We analyze in more detail two connections: *facebook.com* and *redhat.com*. To have more information about the handshake process, we run the *openssl c_client* command passing the options: *-trace -debug -state*. Moreover, we add the options *-tls1_3* to be sure of using TLS 1.3 (as requested), even if the higher protocol should be applied by default in the version negotiation phase. I collected the output in a text file *tls-checkers-deep.txt*, after than I split it into *facebook.txt* and *redhat.txt*.

3.1 ClientHello →

At the beginning the client sends a *ClientHello* message. There is a lot of information here:

- The protocol version *TLS 1.2*, with extension indicating *TLS 1.3*
- A random value (28 bits), plus a timestamp *gmt_unix_time* (4 bits)
- A 32-bits session ID
- The supported cipher suites by the client (in order of preference)
- In the extensions we can find:

  The name of the server

  Supported version (in this case only 1.3 - since we used this option!)

  Supported groups

  Supported signature algorithms

3.2 ServerHello ←

From the server we receive a *ServerHello* message. In its payload we can find:

- The protocol version *TLS 1.2*, with extension indicating *TLS 1.3*
- A new random value (28 bits), plus a timestamp *gmt_unix_time* (4 bits)
- A 32-bits session ID (the same of the client)
- The chosen cipher suite:

  TLS_AES_256_GCM_SHA384 for Red Hat

  TLS_CHACHA20_POLY1305_SHA256 for Facebook
- In the extensions we can find:

  Chosen TLS version: 1.3

  Chosen group: *ecdh_x25519*

  The server ephemeral public key for the key exchange

3.3 Other server messages ←

After that the server sends the *EncryptedExtensions* and the *Certificate* messages. We can see here the certificates containing the servers' public keys.

For Red Hat, having x501 name:

```
C = US, ST = North Carolina, L = Raleigh, O = "Red Hat, Inc.", CN = redhat.com
```

It is signed by:

```
C = US, O = DigiCert Inc, CN = DigiCert Global G2 TLS RSA SHA256 2020 CA1
```

For Facebook we have *CompressedCertificate* in place of *Certificate*, this probably allows it to speed up the handshake. The subject is:

```
subject=C=US, ST=California, L=Menlo Park, O=Meta Platforms, Inc., CN=*.facebook.com
```

and the issuer is:

```
C=US, O=DigiCert Inc, OU=www.digicert.com, CN=DigiCert SHA2 High Assurance Server CA
```

The *CertificateVerify* contains the chain of certificates to verify the certificate.

3.4 Final handshake

Both parties sends each other a *Finished* message, containing the *verify_data* field used to verify and complete the handshake.

The result of the handshake for RedHat is:

```
No client certificate CA names sent
Peer signing digest: SHA256
Peer signature type: RSA-PSS
Server Temp Key: X25519, 253 bits
---
SSL handshake has read 4052 bytes and written 331 bytes
Verification: OK
---
New, TLSv1.3, Cipher is TLS_AES_256_GCM_SHA384
Server public key is 4096 bit
This TLS version forbids renegotiation.
Compression: NONE
Expansion: NONE
No ALPN negotiated
Early data was not sent
Verify return code: 0 (ok)
```

The result of the handshake for Facebook is:

```
No client certificate CA names sent
Peer signing digest: SHA256
Peer signature type: ECDSA
Server Temp Key: X25519, 253 bits
---
SSL handshake has read 2449 bytes and written 317 bytes
Verification: OK
---
New, TLSv1.3, Cipher is TLS_CHACHA20_POLY1305_SHA256
Server public key is 256 bit
This TLS version forbids renegotiation.
Compression: NONE
Expansion: NONE
```

```
No ALPN negotiated
Early data was not sent
Verify return code: 0 (ok)
```

We can noticed that the Facebook's server public key is much shorter, in general less data is needed and the handshake is faster.

4 Classify the results

I used SSL labs to collect security results on the connections. The result can be clustered according to colors: green(A/A+), cyan(B - key ex), yellow(B - protocol), red(F), magenta(T):

Connection	Certificate	Protocol	KeyEx	Ciphers	Overall
google.com	100	70	90	90	**B**
facebook.com	100	70	90	90	**B**
redhat.com	100	100	90	90	**A+**
infinispan.org	100	100	90	90	**A**
almaviva.it	0	70	90	90	**T**
www.eng.it	100	100	90	90	**A+**
edenred.it	100	100	90	90	**A**
esa.int	100	100	90	90	**A**
theguardian.com	100	100	90	90	**A+**
repubblica.it	100	100	90	90	**A**
huffingtonpost.it	100	70	90	90	**B**
quotidianolavoce.it	100	100	90	90	**A**
baraondanews.it	100	70	90	90	**B**
mit.edu	100	100	90	90	**A**
stanford.edu	100	100	70	90	**B**
uniroma1.it	100	100	90	90	**A**
unicusano.it	100	70	90	90	**B**
chiaveorgonica.it	100	70	90	90	**B**
diocesiportosantarufina.it	0	100	90	90	**T**
hwpviewer.hbedu.co.kr	100	70	0	0	**F**

4.1 A/A+ green connections

Those connections are good in every aspect. In particular the A+ ones have also a *DNS Certification Authority Authorization (CAA) Policy* and *HTTP Strict Transport Security (HSTS)* defined. I collected the output in a text file *tls-checkers-deep.txt*

4.2 B yellow connections

Those connections are good on certificate, key exchange and ciphers, while the protocol can be improved on the protocol side. The reason is that those connections support also TLS 1.1 and 1.0 that are now considered weak. According to the tool, even if those versions can be used, the downgrade attack prevention in place. The reason probably is not a lack of skills of the provider of those services, but the strategic choice of support a wider range of clients.

4.3 B cyan connection

In this case certificate, protocol and ciphers are good, while the key exchange process of the TLS handshake can be improved since the server supports the Forward Secrecy only for a few browsers.

4.4 F red connection

This is the case of the connection *hwpviewer.hbedu.co.kr*, as the name suggested it is probably not a production portal. It is nice to analyze it, since we can found very deep vulnerabilities on the key exchange process and on ciphers used.

4.5 T magenta connections

The problem of those connections is the fact that the server certificates have been expired. This is not always true in all cases. From instance in the case of Almamiva portal the certificate is no longer valid only if the connection is served by the **.awsglobalaccelerator.com* host. The *Valid until* certificate field contains the value:

```
Fri, 15 Nov 2024 23:59:59 UTC (expired 1 month and 8 days ago)
EXPIRED
```

In this case the score for the certificate is of course 0.