

I executed a very basic test between kafka-console-consumer and a simple installation of one Kafka broker with TLS.

The test consisted on a Kafka broker with a certificate that didn't match the domain name I used to identify the server. The CA was well set up to avoid related problems, like unknown CA error code. Thus, when the server sends the certificate to the client, the handshake fails with code error 46 (certificate unknown). The goal was that my tool would detect the issue and send an event, describing a TLS handshake problem for both processes.

Many times, while testing my applications, I run into this problem: I run a test without SSL, it works, and when I run it with SSL, with fast created certificates, the client cannot authenticate the server and I end up wasting almost 12 hs trying to find out if it's an application issue or a network issue and what issue. Sometimes because I was too lazy to run tcpdump in a remote server and sometimes even using tcpdump . So I developed this tool that would tell me the problem right away with an event.

However, I noticed the tool sent what I thought it was the wrong event, it sent a TLS exception event for an unexpected message instead of an event for TLS alert for certificate unknown.

I'm using Kafka 2.13-2.8.0, released on April 19th, 2021.

On one side I have a Kafka broker running in node 192.168.56.102, in VirtualBox VM. Listening on port 9093. On the other side I run kafka-console-consumer.sh on node 192.168.56.101, with the following configuration **config/consumer.properties**:

```
ssl.protocol=TLS
security.protocol=SSL
ssl.enabled.protocols=TLSv1.2,TLSv1.1,TLSv1
#ssl.endpoint.identification.algorithm=
ssl.truststore.location=/usr/local/kafka/test/server/kafka.truststore.jks
ssl.truststore.password=wwwwwwwww
ssl.keystore.location=/usr/local/kafka/test/client.keystore.jks
ssl.keystore.password=wwwwwwwww
```

As you can see, I commented out the line for the client not to authenticate the server certificate.

The tool is called DLLT. When I look at the pcap file generated by DLLT,

dump_192.168.56.101_192.168.56.102_32776_9093_2021_10_06_21_09_19.pcap

It is possible to see the zeroes records in Wireshark as Ignored Unknown Records.

Dump_192.168.101.192.168.56.102_32776_9093_2021_10_06_21_09_19.pcap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.56.101	192.168.56.102	TCP	74	32776 → 9093 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=29127911 TSecr=3411998222
2	0.000347	192.168.56.102	192.168.56.101	TCP	74	9093 → 32776 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 SACK_PERM=1 TSval=29127911 TSecr=3411998222
3	0.000366	192.168.56.101	192.168.56.102	TCP	66	32776 → 9093 [ACK] Seq=1 Ack=1 Win=64240 Len=0 TSval=2912792809 TSecr=3411998222
4	0.148778	192.168.56.101	192.168.56.102	TLShv1.2	355	Client Hello
5	0.149209	192.168.56.102	192.168.56.101	TCP	66	9093 → 32776 [ACK] Seq=1 Ack=290 Win=66608 Len=0 TSval=3411998079 TSecr=2912793101
6	0.155926	192.168.56.102	192.168.56.101	TLShv1.2	2398	Server Hello, Certificate, Server Key Exchange, Server Hello Done
7	0.155946	192.168.56.101	192.168.56.102	TCP	66	32776 → 9093 [ACK] Seq=290 Ack=2243 Win=63488 Len=0 TSval=2912792965 TSecr=3411998222
8	0.291422	192.168.56.101	192.168.56.102	TLShv1.2	7306	Client Hello, Ignored Unknown Record
9	0.291447	192.168.56.101	192.168.56.102	TLShv1.2	7306	Ignored Unknown Record
10	0.291772	192.168.56.102	192.168.56.101	TCP	66	9093 → 32776 [ACK] Seq=2243 Ack=7530 Win=81088 Len=0 TSval=3411998221 TSecr=3411998222
11	0.291773	192.168.56.102	192.168.56.101	TCP	66	9093 → 32776 [ACK] Seq=2243 Ack=14770 Win=92672 Len=0 TSval=3411998221 TSecr=3411998222
12	0.291785	192.168.56.101	192.168.56.102	TLShv1.2	2295	Ignored Unknown Record
13	0.292073	192.168.56.102	192.168.56.101	TCP	66	9093 → 32776 [ACK] Seq=2243 Ack=16999 Win=97130 Len=0 TSval=3411998222 TSecr=3411998222
14	0.292075	192.168.56.102	192.168.56.101	TLShv1.2	7306	Server Hello, Certificate, Server Key Exchange, Server Hello Done, Ignored Unknown Record
15	0.292075	192.168.56.102	192.168.56.101	TLShv1.2	7306	Ignored Unknown Record
16	0.292084	192.168.56.101	192.168.56.102	TCP	66	32776 → 9093 [ACK] Seq=16999 Ack=9483 Win=60608 Len=0 TSval=2912793101 TSecr=3411998222
17	0.292094	192.168.56.101	192.168.56.102	TCP	66	32776 → 9093 [ACK] Seq=16999 Ack=16723 Win=55808 Len=0 TSval=2912793101 TSecr=3411998222
18	0.292393	192.168.56.102	192.168.56.101	TLShv1.2	2295	Ignored Unknown Record
19	0.292400	192.168.56.101	192.168.56.102	TCP	66	32776 → 9093 [ACK] Seq=16999 Ack=18952 Win=53888 Len=0 TSval=2912793101 TSecr=3411998222
20	0.292425	192.168.56.101	192.168.56.102	TLShv1.2	73	Alert (Level: Fatal, Description: Certificate Unknown)
21	0.292628	192.168.56.101	192.168.56.102	TCP	66	32776 → 9093 [FIN, ACK] Seq=17006 Ack=18952 Win=53888 Len=0 TSval=2912793101 TSecr=3411998222
22	0.292702	192.168.56.102	192.168.56.101	TCP	66	9093 → 32776 [ACK] Seq=18952 Ack=17006 Win=97130 Len=0 TSval=3411998222 TSecr=3411998222
23	0.293628	192.168.56.102	192.168.56.101	TLShv1.2	73	Alert (Level: Fatal, Description: Unexpected Message)
24	0.293629	192.168.56.102	192.168.56.101	TCP	66	9093 → 32776 [FIN, ACK] Seq=18959 Ack=17007 Win=97130 Len=0 TSval=3411998222 TSecr=3411998222

Frame 9: 7306 bytes on wire (58448 bits), 7306 bytes captured (58448 bits) on interface 0
Ethernet II, Src: PcsCompu_aa:aa:28:13 (08:00:27:aa:28:13), Dst: PcsCompu_e6:6c:24 (08:00:27:e6:6c:24)
Internet Protocol Version 4, Src: 192.168.56.101, Dst: 192.168.56.102
Transmission Control Protocol, Src Port: 32776, Dst Port: 9093, Seq: 7530, Ack: 2243, Len: 7240

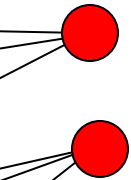
Secure Sockets Layer

- Ignored Unknown Record
 - [Expert Info (Warning/Protocol): Ignored Unknown Record]
 - [Ignored Unknown Record]
 - [Severity level: Warning]
 - [Group: Protocol]

Text item (text). 7240 bytes

Packets: 24 · Displayed: 24 (100.0%) Profile: Default

Source	Destination	Protocol	Length	Info
192.168.56.101	192.168.56.102	TCP	74	32776 → 9093 [SYN] Seq=0 Win=64
192.168.56.102	192.168.56.101	TCP	74	9093 → 32776 [SYN, ACK] Seq=0 /
192.168.56.101	192.168.56.102	TCP	66	32776 → 9093 [ACK] Seq=1 Ack=1
192.168.56.101	192.168.56.102	TLSv1.2	355	Client Hello
192.168.56.102	192.168.56.101	TCP	66	9093 → 32776 [ACK] Seq=1 Ack=29
192.168.56.102	192.168.56.101	TLSv1.2	2308	Server Hello, Certificate, Serv
192.168.56.101	192.168.56.102	TCP	66	32776 → 9093 [ACK] Seq=290 Ack=
192.168.56.101	192.168.56.102	TLSv1.2	7306	Client Hello, Ignored Unknown R
192.168.56.101	192.168.56.102	TLSv1.2	7306	Ignored Unknown Record
192.168.56.102	192.168.56.101	TCP	66	9093 → 32776 [ACK] Seq=2243 Ack
192.168.56.102	192.168.56.101	TCP	66	9093 → 32776 [ACK] Seq=2243 Ack
192.168.56.101	192.168.56.102	TLSv1.2	2295	Ignored Unknown Record
192.168.56.102	192.168.56.101	TCP	66	9093 → 32776 [ACK] Seq=2243 Ack
192.168.56.102	192.168.56.101	TLSv1.2	7306	Server Hello, Certificate, Serv
192.168.56.102	192.168.56.101	TLSv1.2	7306	Ignored Unknown Record
192.168.56.101	192.168.56.102	TCP	66	32776 → 9093 [ACK] Seq=16999 Ac
192.168.56.101	192.168.56.102	TCP	66	32776 → 9093 [ACK] Seq=16999 Ac
192.168.56.102	192.168.56.101	TLSv1.2	2295	Ignored Unknown Record
192.168.56.101	192.168.56.102	TCP	66	32776 → 9093 [ACK] Seq=16999 Ac
192.168.56.101	192.168.56.102	TLSv1.2	73	Alert (Level: Fatal, Descriptio
192.168.56.101	192.168.56.102	TCP	66	32776 → 9093 [FIN, ACK] Seq=176
192.168.56.102	192.168.56.101	TCP	66	9093 → 32776 [ACK] Seq=18952 Ac
192.168.56.102	192.168.56.101	TLSv1.2	73	Alert (Level: Fatal, Descriptio
192.168.56.102	192.168.56.101	TCP	66	9093 → 32776 [FIN, ACK] Seq=189



After sending the zero records, the client sends the correct Alert and then it closes the connection from his side. Then the server sends the Ignored Unknown Records, 16,709 Bytes in total, including the previous messages (Server Hello, ...). Then it sends an Alert for Unexpected Message and closes the connections with a FIN TCP segment.

These TLS messages are due to a coding/design error and don't correspond to the standard TLS handshake.

So, I went to the source code of Kafka to understand what was happening.

Cloned the git repo:

clone <https://github.com/apache/kafka.git>

And worked on Branch **trunk**.

When the server certificate is received, the sequence of calls in SSL stack includes
CertificateMessage.T12CertificateConsumer.checkServerCerts in

share/classes/sun/security/ssl/CertificateMessage.java:

CertificateMessage.T12CertificateConsumer.consume-> onCertificate-> checkServerCerts

The sequence continues:

X509TrustManagerImpl. checkServerTrusted -> checkTrusted-> checkIdentity-> HostnameChecker
.match-> matchDNS

matchDNS checks the common name and it discover it doesn't match, and throws a CertificateMessage
exception:

```
String msg = "No name matching " + expectedName + " found";  
throw new CertificateException(msg);
```

This exception is caught in checkServerCerts that will call TransportContext.fatal(...) and this one logs
what is shown below and returns a SSLHandshakeException with CERTIFICATE_UNKNOWN Alert, that is
thrown by checkServerCerts.

**javax.net.ssl|ERROR|01|main|2021-10-04 07:09:21.828 PDT|TransportContext.java:341|Fatal
(CERTIFICATE_UNKNOWN): No name matching ubuntutests found (**

"throwable" : {

java.security.cert.CertificateException: No name matching ubuntutests found

at java.base/sun.security.util.HostnameChecker.matchDNS(HostnameChecker.java:234)

.....

The SSLHandshakeException is captured in DelegatedTask.run in SSLEngineImpl.

Part of the stack:

at java.base/sun.security.ssl.Alert.createSSLException(Alert.java:131)
at java.base/sun.security.ssl.TransportContext.fatal(TransportContext.java:349)
at java.base/sun.security.ssl.TransportContext.fatal(TransportContext.java:292)
at java.base/sun.security.ssl.TransportContext.fatal(TransportContext.java:287)
at
java.base/sun.security.ssl.CertificateMessage\$T12CertificateConsumer.checkServerCerts(CertificateMes
sage.java:654)
at
java.base/sun.security.ssl.CertificateMessage\$T12CertificateConsumer.onCertificate(CertificateMessage
.java:473)

```
        at
java.base/sun.security.ssl.CertificateMessage$T12CertificateConsumer.consume(CertificateMessage.java:369)

        at java.base/sun.security.ssl.SSLHandshake.consume(SSLHandshake.java:392)
        at java.base/sun.security.ssl.HandshakeContext.dispatch(HandshakeContext.java:443)
        at
java.base/sun.security.ssl.SSLEngineImpl$DelegatedTask$DelegatedAction.run(SSLEngineImpl.java:1074)
    )
        at
java.base/sun.security.ssl.SSLEngineImpl$DelegatedTask$DelegatedAction.run(SSLEngineImpl.java:1061)
    )

        at java.base/java.security.AccessController.doPrivileged(Native Method)
        at java.base/sun.security.ssl.SSLEngineImpl$DelegatedTask.run(SSLEngineImpl.java:1008)
        at
org.apache.kafka.common.network.SslTransportLayer.runDelegatedTasks(SslTransportLayer.java:430)
        at
org.apache.kafka.common.network.SslTransportLayer.handshakeUnwrap(SslTransportLayer.java:514)
        at
org.apache.kafka.common.network.SslTransportLayer.doHandshake(SslTransportLayer.java:368)
            at org.apache.kafka.common.network.SslTransportLayer.handshake(SslTransportLayer.java:291)
            at org.apache.kafka.common.network.KafkaChannel.prepare(KafkaChannel.java:178)
            at org.apache.kafka.common.network.Selector.pollSelectionKeys(Selector.java:543)
            at org.apache.kafka.common.network.Selector.poll(Selector.java:481)
            at org.apache.kafka.clients.NetworkClient.poll(NetworkClient.java:561)
```

The exception is caught as a PrivilegedActionException in SSLEngineImpl.DelegatedTask.run and stored in the handshake context:

hc.delegatedThrown = reportedException

closeReason has been previously set:

closeReason = SSLHandshakeException

In Kafka,

In SslTransportLayer.java

In SslTransportLayer.handshake we call doHandshake which calls handshakeWrap or handshakeUnwrap depending on the case.

After processing the server certificate and the SSSLAuthenticationException is generated in SSLEngineImpl. Eventually method handshakeWrap is called.

handshakeWrap calls

```
SSLEngineResult result = sslEngine.wrap(ByteUtils.EMPTY_BUF, netWriteBuffer);
```

SSLEngineImpl.wrap will call checkTaskThrown and will throw the SSLException it's storing from the previous DelegatedTask executed from SslTransportLayer. handshakeUnwrap.

The SSLException is caught in SslTransportLayer.handshake (SslTransportLayer.java:294)

Then maybeProcessHandshakeFailure is called in the exception handler.

Then handshakeFailure is called from maybeProcessHandshakeFailure

In handshakeFailure releases SSLEngine resources and in line 883 it calls

```
if (!flush || flush(netWriteBuffer))
```

Both ByteBuffers netReadBuffer and netWriteBuffer have size 16,709 (SSLRecord.maxRecordSize)

And in this call to flush

```
netWriteBuffer.position = 0
```

```
netWriteBuffer.limit = 16,709
```

```
netWriteBuffer.capacity = 16,709
```

And this is where the repeated Client Hello record followed with all zeroes is sent to the server before sending the SSL Alert.

Then the SslAuthenticationException("SSL handshake failed ...") is thrown

This exception is capture in

KafkaChannel.prepare as an AuthenticationException and rethrown.

The AuthenticationException is captured in

```
org.apache.kafka.common.network.Selector.pollSelectionKeys()
```

This is printed in Selector.pollSelectionKeys:L616

```
Failed authentication with ubuntu192.168.56.102 (SSL handshake failed)
(org.apache.kafka.common.network.Selector)
```

Then close is called in pollSelectionKeys:

```
close(channel, sendFailed ? CloseMode.NOTIFY_ONLY : CloseMode.GRACEFUL);
```

Going down from this call to close, eventually **SslTransportLayer.close()** is called and the correct TLS Alert is sent. This is printed in SSL logs:

```
> javax.net.ssl|WARNING|01|main|2021-10-04 18:54:17.019
```

```
PDT|SSLEngineOutputRecord.java:168|outbound has closed, ignore outbound application data
```

```
javax.net.ssl|DEBUG|01|main|2021-10-04 18:54:17.021 PDT|SSLEngineOutputRecord.java:505|WRITE:
TLS12 alert, length = 2
```

```
javax.net.ssl|DEBUG|01|main|2021-10-04 18:54:17.028 PDT|SSLEngineOutputRecord.java:523|Raw
write (
```

```
0000: 15 03 03 00 02 02 2E          .....
```

This happens inside SslTransportLayer.close() where sslEngine.wrap is called and then flush(netWriteBuffer);

And here it sends out the correct TLS Alert record.

A similar error can be seen in the server side with a different exception (unexpected message). This is because it received the wrong records and both client and server share the same networking code.