Java 9 security enhancements in practice

Martin Toshev

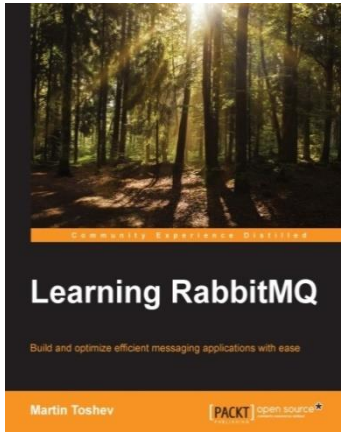
Who am I

Software consultant (CoffeeCupConsulting)

BG JUG board member (http://jug.bg)

OpenJDK and Oracle RBDMS enthusiast

Twitter: @martin_fmi



Agenda

TLS support in JDK

DTLS support in JDK 9

TLS ALPN extension in JDK 9

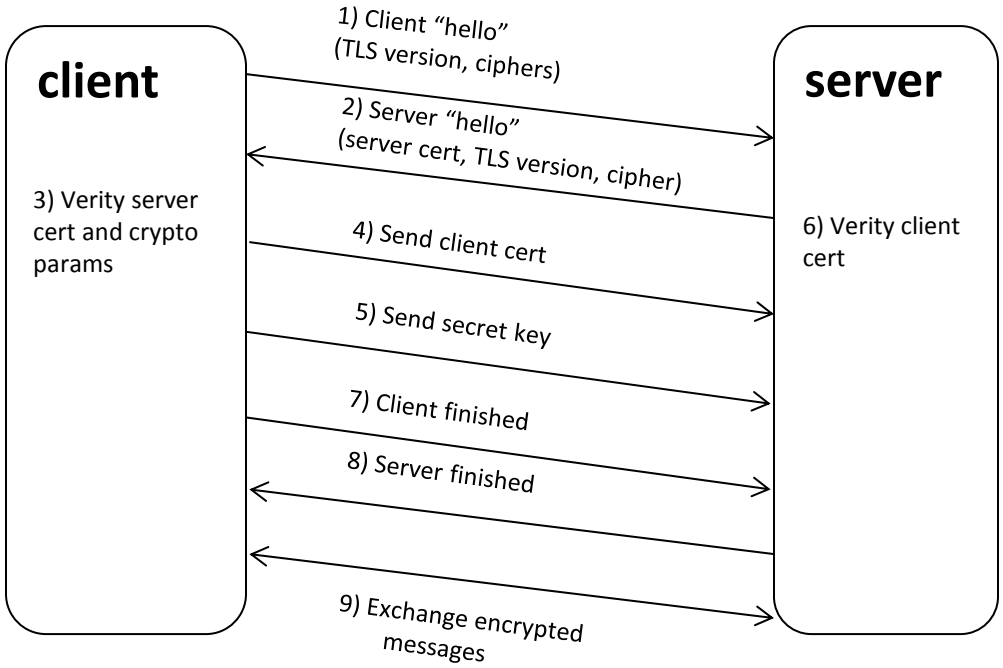
The rest at a glance ...

TLS support in JDK

TLS and the JDK

- Up to JDK 9 TLS 1.0, 1.1 and 1.2 are supported via the JSSE API
- TLS 1.3 specification currently ongoing …
- Typically used to secure most types of application protocols
- Used for the implementation of SSL VPNs

TLS handshake



Java Secure Socket Extension

- Implemented as JCA provider (SunJSSE)
- Core classes part of the **javax.net** and **javax.net.ssl** packages
- Provides APIs for blocking and non-blocking mode of operation
- javax.net.ssl.HttpsURLConnection used to simply HTTPs communication

JSSE blocking mode

- Provided by the **javax.net.ssl.SSLSocket** class
- Used in the same manner as a regular socket
- Handshake might be triggered by:
  - Calling startHandshake() on the socket
  - Calling getSession() on the socket
  - Reading/writing to the socket

JSSE blocking mode (example)

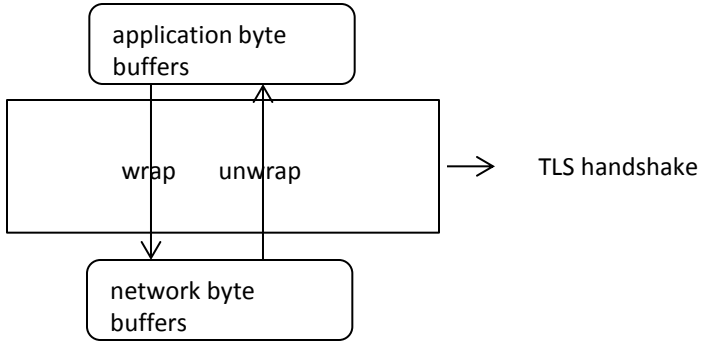
```
System.setProperty("javax.net.ssl.keyStore", "sample.pfx");
System.setProperty("javax.net.ssl.keyStorePassword", "sample");
SSLServerSocketFactory ssf = (SSLServerSocketFactory)
            SSLServerSocketFactory.getDefault();
ServerSocket ss = ssf.createServerSocket(4444);
while (true) {
    Socket s = ss.accept();
    BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()));
    PrintStream out = new PrintStream(s.getOutputStream());
    String line = null;
    while (((line = in.readLine()) != null)) { System.out.println(line); out.println("Hi, client"); }
in.close(); out.close(); s.close();
```


JSSE blocking mode (example)

```
System.setProperty("javax.net.ssl.trustStore", "sample.pfx");
System.setProperty("javax.net.ssl.trustStorePassword", "sample");
SSLSocketFactory ssf = (SSLSocketFactory) SSLSocketFactory.getDefault();
Socket s = ssf.createSocket("127.0.0.1", 4444);
PrintWriter out = new PrintWriter(s.getOutputStream(), true);
out.println("Hi, server.");
BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()));
String x = in.readLine();
System.out.println(x);
out.close(); in.close(); s.close();
```


JSSE non-blocking mode

- Provided by the **javax.net.ssl.SSLEngine**
- The wrap() and unwrap() methods used to transfer data
- Handshake might be triggered by
  - calling beginHandshake()
  - calling wrap()
  - calling unwrap()



JSSE non-blocking mode

- Much more complex to use than the SSLSocket API
- Can be used along with the java.nio.channels.SocketChannel API
- The **javax.net.debug** system property might be very useful for debugging

```
-Djavax.net.debug=all
```

```
-Djavax.net.debug=SSL, handshake
```


DTLS support in JDK 9

DTLS

- TLS over an unreliable transport protocol such as UDP
- Reliable and in-order delivery are not guaranteed
- Targets to secure unreliable protocols such as DNS or SIP etc.
- Follows TLS specifications (hence 1.3 in draft)

DTLS vs TLS

- Added explicit sequence number field
- Dropped support for some ciphers (such as RC4)
- Added retransmission timer for resending of packets
- MAC verification failure triggers warning instead of failure
- Added **HelloVerifyRequest** message in order to identify sender

DTLS before JDK 9

- Pre-JDK 9 a third party provider such as BCJSSE could be used
- … or external libraries such as OpenSSL via JNI

DTLS in JDK 9

- Support for DTLS 1.0 and 1.2
- Implementation adapted to the JSSE API
- SSEngine typically used along with DatagramSocket
- Implementation based on the SSLEngine API
- … which makes it hard to use directly

DTLS in JDK 9

- Ordered delivery provided automatically by SSLEngine
- Sequence number can be retrieved via SSLEngineResult.sequenceNumber()
- Redelivery of failed messages must be done by the application
- … in DTLS handshake messages must be delivered properly

DTLS in JDK 9

```
SSLContext sslContext = SSLContext.getInstance("DTLS");
sslContext.init(…)
SSLEngine engine = sslContext.createSSLEngine();
engine.setUseClientMode(false);
…
```

- Good examples are provided by the JDK 9 test suite

TLS ALPN extension in JDK 9

ALPN

- Used to identify the application protocol during TLS handshake
- Does not require additional roundtrips (ClientHello message used)
- Allows the server to send different certificates for different protocols
- Typical use case is the HTTP 2 protocol
- … as e.g. HTTP 1.1 and HTTP/2 may both reside on the same TLS endpoint

ALPN in JDK 9

- Before handshake set the supported protocols on the socket/engine:

```
SSLParameters sslParams = sslSocket.getSSLParameters();
sslParams.setApplicationProtocols(…)
```

```
SSLParameters sslParams = sslEngine.getSSLParameters();
sslParams.setApplicationProtocols(…)
```


ALPN in JDK 9

- Trigger the handshake on the socket/engine – for example:

```
sslSocket.startHandshake();
```

```
sslEngine.beginHandshake();
```


ALPN in JDK 9

- After handshake you can get the negotiated protocol:

```
String protocol = sslSocket.getApplicationProtocol();
```

```
String protocol = sslEngine.getApplicationProtocol();
```


Advanced ALPN

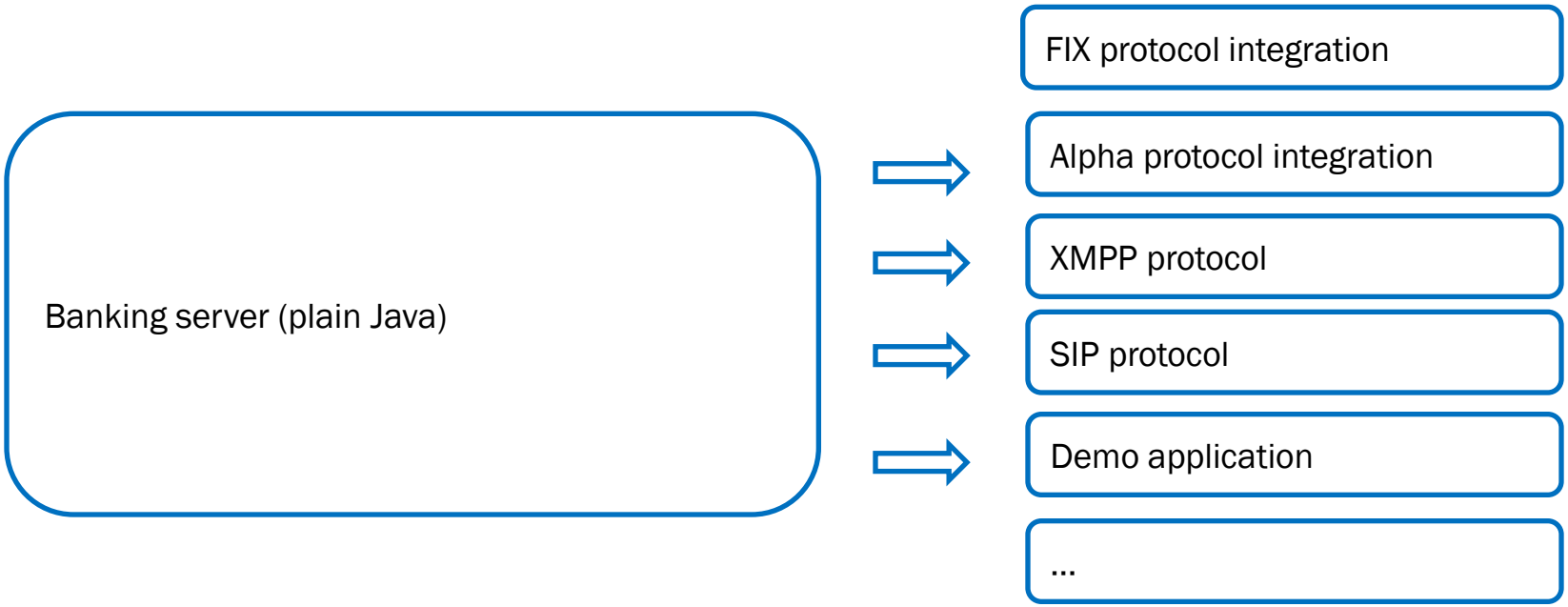
- You can also specify custom protocol resolution

```
sslSocket.setHandshakeApplicationProtocolSelector((serverSocket, clientProtocols) -> {

        SSLSession handshakeSession = serverSocket.getHandshakeSession();
        String cipher = handshakeSession.getCipherSuite();
        int packetBufferSize = handshakeSession.getPacketBufferSize();

        if("RC4".equals(cipher) && packetBufferSize > 1024) {
                return "protocol1";
        } else {
                return "protocol2";
        }
});
```


Demo: banking server



The rest at a glance …

OCSP Stapling for TCP

- Provides a capability for the server to check certificate revocation
- The server typically caches OCSP responses
- Done in order to reduce the number of responses from the OCSP server
- Must be enabled on both the client and the server

```
-Djdk.tls.client.enableStatusRequestExtension=true
-Dcom.sun.net.ssl.checkRevocation=true
```

```
-Djdk.tls.server.enableStatusRequestExtension=true
```


PKCS12 Keystores by default

- PKIX (PKCS12) is default type of store if no other is specified
- Replaces JKS as the default keystore
- PKCS12 provides support for stronger cryptographic algorithms
- Provides better interoperability with other systems

Others

- DRBG-Based SecureRandom Implementations
- Utilization of CPU Instructions for GHASH and RSA
- SHA-1 Certificates disabled for certificate validation
- Implementation of the SHA-3 hash algorithms

Summary

- JDK 9 introduces significant set of security features and enhancements
- The major part of them is related to TLS support
- Hopefully this tendency will continue with future releases …

References

Java Platform, Standard Edition What's New in Oracle JDK 9
https://docs.oracle.com/javase/9/whatsnew/

Java Platform, Standard Edition Security Developer's Guide
https://docs.oracle.com/javase/9/security/

References

JEP 219: Datagram Transport Layer Security
http://openjdk.java.net/jeps/219

JEP 244: TLS Application-Layer Protocol Negotiation Extension
http://openjdk.java.net/jeps/244

JDK 9 SSL test suite
https://github.com/netroby/jdk9-dev/tree/master/jdk/test/javax/net/ssl

References

Bouncy Castle (D)TLS API and JSSE Provider
https://downloads.bouncycastle.org/fips-java/BC-FJA-(D)TLSUserGuide-1.0.0.pdf

Introduction to DTLS
https://www.pixelstech.net/article/1459585203-Introduction-to-DTLS%28Datagram-Transport-Layer-Security%29

DTLS implementation in JDK 9 (changeset)
http://hg.openjdk.java.net/jdk9/jdk9/jdk/rev/6721ff11d592