TLS Implementation for APRS-IS Servers

Peter S. Loveall

ARRL, TAPR, (ISC)²

Author Note

This paper describes the implementation of TLS on javAPRSSrvr.

Abstract

Implementation of TLS encryption in javAPRSSrvr to support HTTPS and Secure WebSocket (WSS) connections. This paper does not address other TCP encryption methods and certificate exchanges experimentally created by other authors for non-HTTPS connections.

Keywords: APRS,TLS,WebSocket,HTTPS

TLS Implementation for APRS-IS Servers

Requirements

HTTP and WebSocket support have been implemented in javAPRSSrvr for over 5 years. HTTP was implemented first to provide an easily implemented interface for submitting single or multiple packets in a send-only format that would be able to utilize HTTP proxies and make use of lower cost HTTP telecommunications available at the time.

WebSocket support was added in 2017 to the standard HTTP send-only listener port to allow clients to connect bidirectionally using proxies and telecommunications facilities which charged less for HTTP communications.

With the advent of public Access Points and man-in-the-middle attacks on non-encrypted data, the need for HTTPS support has become a requirement for almost all websites. This is such a critical piece of security that there are now free SSL certificate vendors and most web hosts support SNI (Server Name Indication) to allow HTTPS to a single IP address with multiple web sites. All modern browsers are moving to HTTPS as the preferred protocol when connecting to a web site and these same browsers refuse to make a non-secure WebSocket connection from a page received via HTTPS. This last fact caused me to consider implementing Secure WebSocket in javAPRSSrvr which, by default, also implements HTTPS.

Project Scope

javAPRSSrvr is a full APRS-IS server implemented 100% in Java and has been in use since 2003. Through the ensuing years, it has constantly been maintained and available to amateur radio operators worldwide. It was the cornerstone in stabilizing APRS-IS by implementing the q algorithm and other loop detection. javAPRSFilter, an adjunct to javAPRSSrvr created by Roger Bille SM5NRK, is the basis for all server filter commands allowing a client to connect to a server and only receive requested packets and any packets necessary to properly operate an IGate. It has been **critical** that any enhancements to javAPRSSrvr do not require any changes to existing APRS-IS communications and user connectivity. This was the basis for implementing WebSockets as well.

This project scope specified implementing a new TCP listener port for HTTPS and implementing a new upstream dialer (connection type) for Secure WebSocket while reusing as much existing code as possible and using Java 8 and later standard constructs and classes. Because javAPRSSrvr was properly developed with little overlap between classes in functionality, this proved to be an addition of a very few lines of code.

A consideration while scoping the project was whether to implement HTTP/2 in addition to the TLS implementations. I determined that HTTP/2 is 100% backward compatible with HTTP/1.1 because its first communication is using HTTP/1.1 to request upgrading to HTTP/2 using the same upgrade mechanism as the WebSocket upgrade request. One of the major advantages of HTTP/2 is using a single connection for multiple requests. Because HTTP send-only ports only support a single request per connection, I determined that it was best to not implement HTTP/2 at this time.

HTTPS Listener Port

Implementation was done by adding code to the existing TCPListenerPort class to create the server socket using javax.net.ssl.SSLServerSocket. Using standard java.net.security classes, a PKCS12 pfx file is used for the server's certificate and the SSLContext created from that file is used to create the SSLServerSocket. I created a properties file which is included in the JAR file that defines the supported TLS protocols and ciphers. This properties file can be overridden with an external file. The included file includes TLSv1.2 (sans CBC algorithm and RSA handshake ciphers) and TLSv1.3. These are supported by all current JVMs 8.0 and later but are nonetheless used only as a filter to already enabled protocols and cipher suites on the server socket making the implementation cross-platform compatible. Because SSLServerSocket is a subclass of ServerSocket, it is used just like any other socket in the TCPListenerPort class, thereby reusing all existing code for actual client connection handling.

The implementation allows a software developer to connect to a HTTPS port using standard HTTPS connection mechanisms available in the software they are using with needing to create extra code. If they were already connecting using HTTP, HTTPS should be a simple change from http:// or ws:// to https:// or wss://.

Upstream Dialer

The upstream dialer uses a class I created called WebSocketClient. When the WebSocket protocol was added in 2017, there was no off-the-shelf classes available so I created both the WebSocketServer and WebSocketClient classes in javAPRSSrvr. This gave me an advantage to implement WSS support by adding it as simply a connection mechanism to existing code. I again went to the javax.net.ssl package available on any Java 8.0 and later JVM and implemented the upstream connection using existing code which can use a proxy (ProxySelector is passed “https” for a WSS connection instead of “http”) if defined to Java and attached a SSLSocket to the socket returned by ProxySelector. That SSLSocket is also restricted to the secure TLSv1.2 ciphers and TLSv1.3 in the same manner as the listener port and it does host name validation on the certificate. Java also supports SNI by default making this upstream dialer compatible with reverse proxies hosted on shared websites.

Summary

Because of the modular design of javAPRSSrvr, upgrading its HTTP and WebSocket support to also support HTTPS and Secure WebSocket using TLSv1.2 and TLSv1.3 was a relatively short requirement to implementation cycle. In addition to testing directly against remote javAPRSSrvr implementations, the upstream dialer has also been successfully tested against a reverse proxy, ARR by Microsoft. As with all development of javAPRSSrvr, it can be used across platforms including its Android derivative, javAPRSSrvrIGate.

The implementation meets the requirements of being additive without affecting current software while providing developers a standards-based (and therefore publicly supported) method of securely connecting to APRS-IS.

References

Overview (Java Platform SE 8) (oracle.com)

RFC 8446 - The Transport Layer Security (TLS) Protocol Version 1.3 (ietf.org)

RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2 (ietf.org)

RFC 6455 - The WebSocket Protocol (ietf.org)

Connecting to APRS-IS