Maxwell Pro TLS Test Suite

Testing TLS Implementations

DevOps, quality assurance engineers and testers use the Maxwell Pro TLS Test Suite to find and fix bugs in their TLS stack or engine. The tests help ensure that the TLS implementation is sufficiently robust so that it is not vulnerable to the wide range of attacks in today's Internet.

The tests make use of the Maxwell Pro network emulation environment, so that each test sequence can intelligently impair all aspects of TLS 1.2, as defined in:

- IETF RFC 5246 “Transport Layer Security, Version 1.2”
- IETF RFC 6176 "Prohibiting Secure Sockets Layer (SSL) Version 2.0”

The TLS Test Suite contains more than 100 unique test cases that take on parameters for greater coverage. The tests ensure TLS 1.2 compliance through vulnerability and robustness testing. The Test Suite supports TLS over TCP/IPv4 or TCP/IPv6.

- Incorporate your own data sources
- No porting exercise to instrument the TLS stack; start testing immediately
- Tech support from senior protocol engineers
- See pass/fail results; no need to analyze complicated outputs
- Replicate customer reported bugs, customize the environment to properly test and replicate the problem

The Maxwell Pro TLS Test Suite provides you with the customization and flexibility to accurately test your implementation. The tests are grouped into categories: Server Handshake, Data Handshake, and Client Handshake, with two traffic sources: "TLS Client (tls)” and “TLS Server (tlss).”

- Set DigestInfo.AlgorithmIdentifier.parameters field for cryptographic hash functions to invalid values.
- Set ProtocolVersion major and minor fields to invalid values in ClientHello.
- Set CBC pad value to invalid lengths.
- Set Handshake length field to invalid lengths.
- Client replaces ClientHello with ServerHello handshake type.
- Client replaces ClientKeyExchange with ServerKeyExchange handshake type.
- Client sends CertificateRequest after ClientHello.
- Set record fragment length field to invalid values.
- Send invalid values during handshake.
- Send Application data record before first ClientHello.
- Set Client.SessionID fields to invalid values.
- Set Client.CipherSuite fields to invalid values.
- Set Client.CompressionMethod fields to invalid values.
- Change the ChangeCipherSpec message's type to various values.
- Set HandshakeType of finished type to invalid value.
- Set ClientHello.extensions length so that it extends past end of message.
- Manipulate SignatureAndHashAlgorithm.hash value to various values.
- Change the first SignatureAndHashAlgorithm.signature value to various values.
- Do not send a Certificate when the server sends a CertificateRequest.
- Set ClientHello.random.gmt_unix_time to various values.
- Manipulate padding and padding_length bytes of block cipher.
- Send a Client Certificate when a server has not requested one.
- Do not send the Client Key Exchange message.

- Negotiate RSA key exchange without signature_algorithms extension.
- Negotiate DSS key exchange without signature_algorithms extension.
- Test that the server sends ServerKeyExchange for DHE_DSS key exchange.
- Test that the server sends ServerKeyExchange for DHE_RSA key exchange.
- Test that the server sends ServerKeyExchange for DH_anon key exchange.
- Test that the server doesn't send ServerKeyExchange for RSA key exchange.
- Send an RSA-Encrypted Premaster Secret Message that cannot be processed.
- Send Finished message immediately after ClientHello sent.
- Manipulate fields and values of ClientHello.extensions.
- Compute PRF in Client Finished message using an incorrect finished_label.
- Verify implementation of TLS_RSA_WITH_AES_128_CBC_SHA.
- Create compressed record that decompresses to various values.
- Test that the server implements TLS_RSA_WITH_NULL_SHA256.
- Test that the server implements TLS_RSA_WITH_AES_128_CBC_SHA256.
- Test that the server implements TLS_RSA_WITH_AES_256_CBC_SHA256.
- Test that the server implements TLS_DHE_DSS_WITH_AES_128_CBC_SHA256.
- Test that the server implements TLS_DHE_RSA_WITH_AES_128_CBC_SHA256.
- Test that the server implements TLS_DH_anon_WITH_AES_128_CBC_SHA256.
- Test that the server implements TLS_DHE_DSS_WITH_AES_256_CBC_SHA256.
- Test that the server implements TLS_DHE_RSA_WITH_AES_256_CBC_SHA256.
- Test that the server implements TLS_DH_anon_WITH_AES_256_CBC_SHA256.
- Reuse SessionID but exclude previous cipher suite from ClientHello.
- Reuse SessionID but exclude previous compression method from ClientHello.
- Renegotiate handshake only up to but not including ChangeCipherSpec.
- Manipulate ContentType with invalid values.
- Introduce max and min TLSPlaintext fragments.
- Send various types of Alerts, both valid and invalid, at different points in the handshake.
- Set record fragment length field to zero for ChangeCipherSpec types.
- Set TLSCiphertest.length one longer than actual block length.
- Server replaces ServerHello with ClientHello handshake type.
- Client replaces ServerKeyExchange with ClientKeyExchange handshake type.
- Introduce invalid operations and values with ServerHello.
- Introduce invalid operations and values with Server.SessionID.
- Introduce invalid operations and values with Certificate.certificate_list.
- Compute PRF in Server Finished message using an incorrect finished_label.
- Verify implementation of TLS_RSA_WITH_AES_128_CBC_SHA.
- Send signature_algorithms extension to TLS client
- Manipulate fields and values of ServerKeyExchange message.
- As anonymous server, issue various invalid requests
- Verify that a TLS server does not accept downgrade to SSLv3.*
- Verify that a TLS server does not accept zero padding in block ciphers.*
- Verify that a TLS client does not accept downgrade to SSLv3.*
- Verify that a TLS client does not accept zero padding in block ciphers.*
- Verify that a TLS client does not accept non-standard padding in block ciphers.*

*The POODLE attack relies on negotiating a downgrade to SSLv3 and also on use of non-standard pad values in block ciphers, so these tests were added per RFC requirements and security community suggestions.

TLS Client and Server Side Test Coverage

Maxwell Pro supports **both** server side and client side TLS testing. Server side testing is possible straight "out of the box" with no programming needed. Client side testing requires automating repeated initiations of client connections from the target test machine to Maxwell Pro.

After the initial connection handshake, the TLS protocol dialog for data transfer is effectively identical for the client and server side. Thus, only a small fraction of any tests from any supplier are relevant to the data transfer phase. The Maxwell Pro tests provide 100% dialog coverage, and the most complex portion, the handshake, can always be tested with no extra porting or programming effort. You can just use an existing TLS client or server application on your target test machine. Only a couple of the data transfer tests may benefit from writing a TLS echo service program for the target test machine.

Establishing a source of authority

The Maxwell Pro TLS Test Suite references the RFCs that correlate to each test area. These official IETF documents detail the Internet standards and best current practices that can point the user toward a better understanding of the problem.

Sample Test Documentation...

PURPOSE OF THE TEST

Make the length field of the first extension in ClientHello.extensions invalid.

WHAT THE TEST DOES

The length field of the first extension in the ClientHello.extensions list is set to 2^15-1, which will exceed the Handshake. length field.

ASSUMPTIONS

None.

EXPECTED OUTCOME

DUT should issue an illegal parameter alert and must close the connection.

RELEVANT TRAFFIC SOURCES

tls.v4, tls.v6

REFERENCES

RFC 5246, Page 37, 'struct ... Handshake', page 41, 'extensions'.'

Please Contact:

Kings Village Center #66190
Scotts Valley, CA 95067
iwl.com
+1.831.460.7010
info@iwl.com