Transport Layer Security Working Group
INTERNET-DRAFT
Expires May 31, 1997

Tim Dierks
Consensus Development
November 26, 1996

Modifications to the SSL protocol for TLS
<draft-ietf-tls-ssl-mods-00.txt>

Status of this memo

Abstract

This document recommends for several modifications be made to the SSL 3.0 **protocol as it is standardized by the IETF under the name of TLS.** These changes primarily standardize various technical details of the protocol and make some other minor modifications.

1. MAC algorithm

SSL 3.0 uses a version of the HMAC algorithm which is not the most recent or the recommended standard. The SSL MAC is defined as:

```
    hash(MAC_secret + pad_2 +
         hash(MAC_secret + pad_1 + data));
```

Where data is the concatenation of several record header fields and the record data. pad_1 and pad_2 are repetitions of the value 0x36 and 0x5C, respectively. These bytes are repeated a number of times dependent on which hash algorithm is being used: 48 times for MD5 and 40 times for SHA.

I recommend moving to HMAC as described in <draft-ietf-ipsec-hmac-md5-01.txt> by incorporating this algorithm into the TLS standard (or by referring to it, should it become an RFC

standard). This formulation uses a 64 byte pad which is combined with the MAC secret using XOR rather than concatenation.

This would replace the MAC formulations used in the certificate verify and finished messages (where the MAC_secret is the master secret) and the MAC formulation used in the record layer (where the MAC_secret is generated specifically for each connection direction.)

2. MAC contents

Currently, the SSL record layer applies the MAC to every element in the record except for the protocol version encoded into every packet. It is inappropriate to transmit values which might affect the functionality of the connection without applying the MAC to these values. If the version number does not control the function of the channel, it should be eliminated; if it does affect the communication, it should be MACed. Thus, I recommend that the data which is MAC`d be amended to:

```
seq_num + SSLCompressed.type + SSLCompressed.version +
    SSLCompressed.length + SSLCompressed.fragment
```

3. Block padding

Padding is required when working with block ciphers to expand source data to an even multiple of the block length. SSL specifies padding, but does not specify a particular value. In order to ensure that implementors do not accidentally transmit unintended data in uninitialized padding fields, I recommend that the TLS add a requirement that the padding be initialized to a particular value. I propose that the padding field must be zeroed and that implementations should check for appropriate padding on incoming records.

4. Message order standardization

In the original SSL 3.0 specification, an error made the statement of when the certificate request message should be transmitted unclear, and different implementations send it in two places: either before or after the server key exchange message. I propose that for the TLS specification, the certificate request message be clearly specified to follow the server key exchange message.

5. Certificate chain contents

In the original SSL 3.0 specification, the text required that a complete X.509 **certificate chain be sent up to and including the self-signed root** cert. It is claimed that this was not the intent of the drafters, and in fact, many implementations do not comply with this portion of the standard. Thus, I propose that the TLS specification clearly state that a partial certificate chain is acceptable if it can be reasonably hoped that the peer will hold all needed certificates to complete the chain.

6. The no_certificate alert

The no_certificate alert, which is to be sent by a client which does not have a suitable certificate to provide a server, presents a subtle problem to the SSL implementer. Because the message order of the SSL protocol is for the most part well defined and enforced, what messages have arrived is very important to the state machine which manages the handshake protocol. Because this alert can replace a handshake message, the alert protocol must communicate to the handshake protocol that this alert has arrived. This is the only place where such a piece of promiscuity is required, thus I recommend that in place of sending a no_certificate alert, TLS clients who do not have a suitable certificate for a server submit instead an Certificate message which contains no certificates.

7. Additional alerts

SSL doesn`t have a great deal of variety in its error alerts. I propose that the following alerts be added to the specification:

internal_error [fatal]: an internal error unrelated to the peer or the correctness of the protocol makes it impossible to continue (such as a memory allocation failure).

user_canceled [fatal]: the user aborted this handshake or connection for some reason.

decrypt_error [fatal]: a public or private key operation failed due to using the wrong key

decode_error [fatal]: a message could not be decoded because some field was out of the specified range or the length of the message was incorrect.

export_restriction [fatal]: an attempt to circumvent export restrictions was detected; for example, attemption to transfer a 1024 bit ephemeral RSA key for the RSA_EXPORT handshake method.

protocol_version [fatal]: the protocol version the peer has attempted to negotiate is recognized, but not supported. (For example, old protocol versions might be avoided for security reasons).

record_overflow [fatal]: an SSLCiphertext record was received which had a length more than 2^14+2048 bytes, or a record decrypted to a SSLCompressed record with more than 2^14+1024 bytes.

decryption_failed [fatal]: a SSLCiphertext decrypted in an invalid way: either it wasn`t an even multiple of the block length or its padding values, when checked, weren`t correct.

access_denied [fatal]: a valid certificate was received, but it did not pass the access control mechanism.

unknown_ca [fatal]: a valid certificate chain or partial chain was received, but the certificate was not accepted because the CA certificate could not be located or couldn`t be matched with a known, trusted CA.

insufficient_security [fatal]: returned instead of handshake_failure

when a negotiation has failed specifically because one of the parties
requires ciphers more secure than those supported by their peer.

    no_renegotiation [warning]: generated in response to a hello request

or a client hello sent on an already negotiated channel. This informs the requestor that no response will be generated, as this entity does not want to renegotiate security parameters (as you might wish to do if there` no way to communicate security parameters up the stack to the client after initial negotiation.

8. Seperation of Record and Handshake layers

The SSL Record Protocol and Handshake Protocol can be viewed as two independant layered protocols: the Record Protocol provides encrypted, reliable transport, and the Handshake Protocol provides algorithm and key negotiation and peer authentication. I propose that they be formally seperated into two documents, or at least two distinct sections of the TLS document. This should make their interoperation clearer, aiding security analysis and perhaps allowing utilization of the Record Protocol with some other handshake protocol or vice-versa.

9. Additional Record Protocol clients

The SSL Record Protocol supports transmitting many different kinds of records over a single connection. This is already used for distinguishing different kinds of protocol messages from each other and from application data. I propose that TLS clearly specify that layered protocols are allowed to use the Record Protocol to transport new record types.