Differences Between SSLv2, SSLv3, and TLS

Loren Weith: 0600978

July 3, 2006

SSLv2, SSLv3, and TLS (1.0) all provide for a secure channel between clients and servers: if looked at in terms of the OSI reference model, SSL and TLS are said to be at the Presentation Layer (layer 6.) SSLv3 contains improvements to SSLv2 and TLS[3] is almost exactly like SSLv3 but it is the outcome of the IETF standardization process for SSLv3. They are apparently so similar that the specification for the TLS versions of IMAP, POP3, and ACAP says in it's first line refers to TLS as "the TLS protocol (formerly known as SSL)"[8].

As the name SSL (Secure Socket Layer) implies, SSL was meant to work very similarly to Berkeley sockets so that applications that were initially designed to use the sockets interface could be easily ported. Generic SSL wrappers like stunnel[1] can be used to directly wrap unmodified servers for certain protocols (such as POP3 and IMAP) if the only difference between the "s" version of the protocol and the original is the encryption and port number (the TLS session is expected on connect – often referred to as tls-on-connect.) In fact, the RFC describing the TLS extensions of POP3 and IMAP only grudgingly support the tls-on-connect method and the RFC recommends against it[8].

Many of the TLS'ified protocols extend the existing protocol to support a `STARTTLS` command that initiates TLS if supported by the client, thus allowing client software developers to add the capability without requiring their users to make configuration changes in order to benefit from better security (a weak argument since the software could just check for the existence of the s-protocol port and then fall back) and it also keeps aging firewalls from preventing secure communication over older protocols like SMTP. Some TLS extensions like the TLS extension to SMTP intentionally don't allow tls-on-connect at all.

1 SSLv2

1.1 Protocol Overview

The SSLv2 protocol can be broken down into the handshake and data transfer modes.

1.1.1 Handshake

The handshake protocol comes first and establishes the security parameters of a connection that may or may not be part of an existing session. During the course of the protocol, either the identity of the server or both the client and server can be verified using X.509 certificates. The master encryption key can be shared across multiple connections to avoid unnecessarily repeating the fairly expensive key negotiation process any more than is necessary.

Presumably the motivation for the feature came from the design of HTTP 1.0, which does not support multiple operations in a single server connection as HTTP/1.1 does, forcing browsers to make tons of connections to render a single page. Browsers responded by making several concurrent connections to web servers to mitigate the performance hit of opening connections as needed (latency kills here, not bandwidth so sending a stack of SYN's up front, then multiple ACK's, etc. is substantially faster than executing multiple consecutive setups.) Adding RSA to the mix would have limited the effectiveness of that strategy.

The following is a brief synopsis of the handshake protocol for SSLv2:

- **No existing session**[7]: Since there is no existing session identifier, there is no session and one needs to be built up. The client issues a challenge, the server chooses a connection-id, then the client verifies the server's identity and then chooses a session key and security spec for the session based on an advertisement of the server's capabilities that is unauthenticated. The session key is the output of a `MD5Hash( master-key, [0 or 1]`[1]`, challenge, connection-id)` so it should be different for every connection and direction. The server can then either send back the challenge and generate a session-id (encrypted and authenticated) or it can ask for and verify a client certificate before doing so.
- **Client and server have a common session identifier**[7]: This version runs similarly to above except that the client includes a session-id in the initial communication and if the server knows about it then they skip the key agreement phase since it has already happened.

1.1.2 Records

Every message in an SSLv2 exchange is embedded in a record and a record is made up of a record header and payload, whether encrypted or in the clear. Because SSLv2 sometimes uses block ciphers, the data to be encrypted must be a multiple of the block size for the encryption algorithm so padding may be added to force the input to the encryption algorithms to be the right size. The cleartext payload of the record is then padded out, encrypted on the appropriate session key, and placed after the record header.

The record header is a 16 or 24 bit string indicating the length of the record and the length of the padding (if any padding needed.) The first two bytes

[1] 0 for client-read-key, 1 for client-write-key

indicate whether there is padding or a "security escape" and the length of the actual data portion of the record. The possible third byte indicates the padding length.

The first bit of the string indicates whether there is padding (0 if there is, 1 if there isn't) and the second bit indicates the presence of a "security escape"[2], the next 14 bits are the size of the data in the record, and the last byte (if there is padding) is the length of the padding[9]. A connection is composed of multiple records. Each transmission in the protocol has a sequence number that is protected by a message authentication code, protecting the stream from a replay attack[7].

1.2 Security Problems

- **The client chooses a cipher spec based on information provided by the server that is not authenticated**. An attacker can thus influence the choice of cipher-spec to something that is supported but easier to crack than the client's choice would have been[2]. Wagner and Schneier give the example of one of the export ciphers, though neither Apache nor Firefox actually support the export crypto option[10].

- **Truncation attacks are possible** because the end of an SSLv2 session is merely signalled by tearing down the underlying TCP session[2]. On the other hand, some protocols running on top of SSLv2 have structures in place to detect truncated transfers (for example, the recommended `Content-length` header in HTTP)[5] can be used to detect such truncations.

- **The same key used for encryption is used for message authentication** so if an exportable browser is used, assurance of authenticity is weakened to the same degree as privacy of the data, which is apparently not required to meet US export regulations. SSLv2 only supports the use of MD5, which is known, at least today, to have some serious flaws[2][11].

- **Padding size is included in the cleartext record header unnecessarily**. SSLv2 does not support padding except in the case of block ciphers where it is needed to make the cipher work. Even then, the padding size is in the cleartext portion of the record header. The record protocol needs only the length of the total record to be cleartext in order to distinguish individual records. Having the exact data length in clear text gives an adversary better traffic analysis capability. It is enough that the record size be a multiple of the chosen cipher's block size: the pad size should be placed in the payload so that it gets encrypted with the rest of the payload to hide the size of the actual data better. It would also be nice to allow a wider range of pad sizes so that random padding can be used to avoid giving hints about the type of traffic going across the connection[10].

- **Data are not compressed before encryption**: a cryptanalyst is aided by knowing something about the structure of the plaintext so it is to

[2] no security escapes are defined in version SSLv2

our advantage to deny the cryptanalyst that information. Compression eliminates most structure and redundancy from the plaintext and usually makes it smaller, limiting a cryptanalyst's knowledge of the plaintext's structure and improving transmission speed for free.

2 SSLv3 vs SSLv2

SSLv3 is said to be a significant improvement over SSLv2 in many ways and many of those will be enumerated here.

2.1 Differences

- The SSLv3 standard supports fall-back to SSLv2 so the obvious attack would be to make the two sides talk to each other using SSLv2 and attack that way. The compliant SSLv3 implementations prevent v3-capable clients and servers from talking to each other using the v2 protocol by using non-random PKCS padding (the first 8 bytes are 0x03) for the RSA encryption of the key data. SSLv3 partners will notice this and refuse to play, v2 partners will not notice and will play, thus no problem. The rest of the padding is random so there is minimal impact on the security added by random padding[6].
- **support for anonymous connections**: SSLv3 has three modes of key exchange: *authenticated server*, *authenticated server and client*, and *anonymous*. Anonymous key-exchange is new and inherently susceptible to man in the middle attacks since there is no certifying authority signature to use for verification of the RSA server RSA key[6].
- **Protection against outside manipulation of handshake protocol**: The finish messages are authenticated and include a hash of all the previous messages including the finish message so that the whole transaction will fail if there is any difference between what each party though happened and what happened. It's a nice solution, though it does allow almost the whole transaction to take place before calling a stop to it[6][10].
- **Truncation attacks are not possible in SSLv3**: closure alerts[3] were added so that the connection only closes without error if a closure message has been received (either end can initiate). It is also allowed for one of the parties to send the close notify and then disconnect without waiting for a response back. The session becomes un-resumable if a connection shuts down unexpectedly[6]. There is no mention of passing the fact that the session ended abnormally to the layers above but that is likely to be done in actual implementations.
- **Compression is required by SSLv3** although the compression algorithm starts out being the CompressionMethod.null (identity function)[6]. Any good implementation will presumably insist on using something better.

[3] such alerts are authenticated

- **Diffie-Hellman key exchange** is a possible choice in SSLv3 and man in the middle attacks have been prevented by authenticated exponents[10].
- **In most situations where hashes are used, both MD5 and SHA-1 are used** to prevent the failure of one of them to completely destroy the security of TLS.[6]

2.2 Problems with SSLv3

- **It is possible to cause a key exchange algorithm rollback** if the SSL connection is only attempting to provide authenticity services and not encryption[10].
- **Replay attacks on anonymous key exchange**: That allows an attacker to pretend to be the server without detection but of course there is not much that can be accomplished with that since he can't guess the private key associated with the random RSA key being talked about.
- **Known plaintexts are available** to an attacker in certain portions of the protocol[10]. Of course the cryptosystems in use are supposed to be resistant to such attacks and that is likely to be the reason why the designers were not terribly concerned. On the other hand, if they are not necessary to be revealed they should not be.
- **Ad-hoc use of message authentication codes**: there are MAC constructions that have not been subject to analysis such as HMAC[10]. Wagner and Schneier don't have any specific objections to them but the point is valid: it's better to use something that has withstood a fair amount of analysis than something that hasn't, particularly in fielded systems.

3 SSLv3 vs TLS 1.0

The TLS specification is very close to SSLv3, right down to section numbers and identical wording in many cases. The differences include the following[2]:

3.1 Differences

- Ad-hoc MAC's were changed to HMAC's as suggested by [10][2].
- Added protocol and cipher suite requirements: Diffie-Hellman, Digital Signature Standard (DSS), and Trible-DES are required instead of optional[2].
- Changes were made to client-write and server-write key calculation [2] claims that the changes are an improvement which it may well be.

3.2 Problems

Any problems with SSLv3 that were not mentioned in the list of differences are likely also to be problems in TLS 1.0.

- A chosen plaintext attack exists that requires knowledge of the initialization vector for a record.[4]

4 TLS 1.1 vs TLS 1.0

4.1 Differences

- TLS 1.1 no longer requires sessions to be restarted in the event of an unexpected connection drop.[4]
- The export-restriction alert is no longer used.[4]
- Certificate format change to PKCS#1[4]
- Export grade cryptography was removed and TLS 1.1 implementations must not support them any longer.[4]
- Explicit initialization vectors are used to address a known plaintext attack in TLS 1.0.[4]
- "Handling of padding errors changed from bad-record-mac rather than decryption-failed alert to protect against CBC attacks"[4]
- "IANA registries are defined for protocol parameters"[4]
- Informational notes about new attacks on TLS added.[4]

References

[1] Brian Hatch. Stunnel – Universal SSL Wrapper. `http://www.stunnel.org/`.

[2] EADS Defence and Security Systems SA. Investigations about ssl. `http://www.eucybervote.org/Reports/MSI-WP2-D7V1-V1.0-02.htm`.

[3] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246 (Proposed Standard), January 1999. Obsoleted by RFC 4346, updated by RFC 3546.

[4] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346 (Proposed Standard), April 2006. Updated by RFC 4366.

[5] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFC 2817.

[6] Alan O. Freier, Philip Karlton, and Paul C. Kocher. The ssl protocol version 3.0. http://wp.netscape.com/eng/ssl3/draft302.txt.

[7] Kipp E. B. Hickman. SSL 2.0 Protocol Specification. http://www.netscape.com/eng/security/SSL_2.html.

[8] C. Newman. Using TLS with IMAP, POP3 and ACAP. RFC 2595 (Proposed Standard), June 1999.

[9] Adam Shostack. An overview of ssl (version 2), May 1995. http://www.homeport.org/~adam/ssl.html.

[10] David Wagner and Bruce Schneier. Analysis of the ssl 3.0 protocol. In *2nd USENIX Workshop on Electronic Commerce*, November 1996.

[11] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu. Collisions for hash functions md4, md5, haval-128 and ripemd. Cryptology ePrint Archive, Report 2004/199, 2004. http://eprint.iacr.org/.