TLS Working Group	Y. Nir
Internet-Draft	Check Point
Intended status: Standards Track	September 25, 2011
Expires: March 28, 2012	

A Method for Sharing Record Protocol Keys with a Middlebox in TLS
draft-nir-tls-keyshare-01

Abstract

This document contains a straw man proposal for a method for sharing symmetric session keys between a TLS client and a middlebox, so that the middlebox can decrypt the TLS-protected traffic.
This method is an alternative to the middlebox becoming a proxy.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

TLS ([TLS]) is used in a wide variety of protocols. The most common use is for protecting HTTP, as described in [HTTPS]. Middleboxes such as firewalls scan protocols for attacks. For HTTP common attacks to scan for are cross-site scripting and transfer of files containing malware. TLS provides authentication and privacy against eavesdropping, but it hides the traffic not only from mallicious intercepters. It also hides the traffic from the middlebox, and prevents it from doing its job. Our goal is to allow the middlebox to inspect the traffic, without allowing others to do the same.
The requirements can be summed up in the following points:

* The middlebox should be able to decrypt all TLS traffic, and optionally (the client's option) also modify it.

* The protocol must not make it easier for other entities to decrypt the traffic.

* The client should be able to opt out of TLS decryption, but opting out may mean that the connection is blocked.

* The server should be able to opt out of TLS decryption, but opting out may mean that the connection is blocked.

Two proposals have been offered to achieve these goals. One is having the middlebox be a proxy, acting as server to the client, and as a client to the server. This option is implemented in several commercial products. [proxy_server_ext] describes an extension to TLS for improving that mechanism, and also contains a good description in the introduction.
This document describes an alternative mechanism, where the client sends the keys to the middlebox in the TLS record stream. This requires more changes to clients and servers, but has the advantage that it does not break many of TLS guarantees.

1.1. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Protocol Overview

A supporting client will send a new extension in the ClientHello message. This new extension is called tls_keyshare. A server that supports this extension will send the extension in the ServerHello if it has received that extension in the ClientHello. Note that sending this extension only acknowledges understanding the protocol, not agreement to decryption. The extension contains a sequence of SHA-256 hashes of middlebox certificates. The client sends the hashes of the certificates of middleboxes that it knows are on-path to the server. See Section 4 for a discussion of middlebox discovery. The server sends a subset of the same hashes, only those for which it agrees to decryption.
This document defines a new record type called KeyshareInfo. This is a new content type rather than a new handshake message so that it doesn't figure in hash calculation of the hash message. A middlebox inserts a KeyShareInfo record into the server-to-client stream immediately after receiving the ClientHello message, if its hash was not present in the client's tls_keyshare extension. It contains two pieces of information:

* A certificate of the middlebox. The public key in the certificate MUST be of the RSA type. The certificate should contain enough information for the client to recognize the middlebox.

* A signature using the private key associated with the certificate over the concatenation of the ClientHello and ServerHello messages.

The middlebox inserts a KeyShareInfo record with a certificate into the client-to-server stream without an alert, immediately following a ServerHello message that does not contain the middlebox hash. The server will reply with either a fatal UNAUTHORIZED_MIDDLEBOX alert, or a fatal RETRY_MIDDLEBOX alert, depending on policy.
In cases where the client and server negotiate either a ciphersuite that the middlebox does not support, or an extension that it doesn't support, the middlebox inserts a different kind of KeyShareInfo record into the stream, that identifies the unsupported ciphersuite or extension. Both kinds of KeyShareInfo records are followed by a fatal alert. The client is expected to add the hashes and remove the unsupported ciphersuites and extensions, before attempting a new TLS connection.
The client inserts a third type of KeyShareInfo record into the client-to-server stream immediately following the ChangeCipherSpec record (before the Finished handshake record). This KeyShareInfo record is constructed differently, and contains an RSA encrypted record of the write keys for both client and server. The client may send several records if there is more than one middlebox.

```
  Client                       Middlebox                          Server
  ------                       ---------                          ------
  ClientHello(tls_keyshare=0)
                                   -------->
                              KeyShareInfo(cert,sig)
                              KeyShareInfo(reject cipher:0x0044)
                              alert(MIDDLEBOX_PRESENT)
                                    <--------
  ClientHello(tls_keyshare=cert_hash)
                                   -------->
                                                    ServerHello(cert_hash)
                                                             (Certificate)
                                                       (ServerKeyExchange)
                                                           ServerHelloDone
                                   <--------
  (Certificate)
  ClientKeyExchange
  (CertificateVerify)
  ChangeCipherSpec
  KeyShareInfo(keys)
  Finished                         -------->
                                                          ChangeCipherSpec
                                                                  Finished
                                   <--------
```

The diagram below outlines discovery.

```
   Client                      Middlebox                        Server
   ------                      ---------                        ------

   ClientHello(tls_keyshare=cert_hash)
                                   -------->
                                                  ServerHello(keyshare=0)
                                                            (Certificate)
                                                      (ServerKeyExchange)
                                                          ServerHelloDone
                                   <--------
                              KeyShareInfo(cert,sig)
                                   -------->
                                             alert(UNAUTHORIZED_MIDDLEBOX)
                                   <--------
```

The diagram below outlines the protocol in a case where the server refuses decryption.

2.1. The tls_keyshare Extension

The tls_keyshare extension is a ClientHello and ServerHello extension as defined in section 2.3 of [TLS-EXT]. The extension_type field is TBA by IANA. The format is to be added.

2.2. The KeyShareInfo Record

The format of the KeyShareInfo record is to be added. The content type is TBA by IANA.

2.2.1. The KeyShareInfo Discovery Subtype

The KeyShareInfo Discovery record gives client or server information about the middlebox. Format is TBA.

2.2.2. The KeyShareInfo Rejection Subtype

The KeyShareInfo Rejection record gives client a list of unsupported ciphersuites and extensions. Format is TBA.

2.2.3. The KeyShareInfo Keys Subtype

The KeyShareInfo Keys record is send by the client to the middlebox and includes the session keys. Format is TBA.

3. Processing

3.1. Client Processing

If the client policy prohibits decryption, the client SHOULD send the tls_keyshare extension without hashes. Note that the middlebox might

still try to proxy the connection, but that is in conflict with this specification, and is outside the scope of this document.
If there are some middleboxes that are by policy acceptable to the client, their certificates are known in advance, and the client believes that they are on-path to the server, then the client MUST send the SHA-256 hashes of their certificates in the tls_keyshare extension.
If a KeyShareInfo Discovery record is received with an unknown certificate, it MAY be ignored, or the user MAY be prompted to authorize the decryption, and optionally change the configuration to allow future decryption by this certificate. There will certainly be controversy about this, but the configuration must happen an some point.
If policy dictates that the particular middlebox referenced in the KeyShareInfo record is not allowed to decrypt, then such a record MUST be ignored. In that case the connection fails. If the middlebox is acceptable, then the client retries the connection, this time adding the SHA-256 hash of the certificate to the tls_keyshare extension. This is the discovery mechanism.
For all the middleboxes that are not ignored, the client MUST send a KeyShareInfo record with the symmetric keys immediately following the ChangeCipherSpec record before any protected record is sent.
If a KeyShareInfo Rejection record is received, the client SHOULD retry the handshake, this time without the flagged ciphersuites and extensions. If it is not acceptable to run the connection without these ciphersuites or extensions, the client should log the event or inform the user.
If the server sends a RETRY_MIDDLEBOX alert, the client should retry the handshake. If it sends an UNAUTHORIZED_MIDDLEBOX alert, then the client should log the event or alert the user.

3.2. Server Processing

The server SHOULD send the tls_keyshare extension even if policy dictates that the decryption is prohibited. If policy allows all middleboxex to decrypt, it makes sense to simply copy the client's tls_keyshare extension.
If some of the middlebox hashes included in the client's tls_keyshare extension are recognized as those of acceptable middleboxes, then only those are copied to the server's tls_keyshare extension. When the middlebox sends a KeyShareInfo Discovery record, the server may decide whether that is acceptable or not, and accordingly send the RETRY_MIDDLEBOX or UNAUTHORIZED_MIDDLEBOX alerts. In any case, every time the server does not copy all hashes from the client's tls_keyshare, the connection is probably going to end in an alert.

3.3. Middlebox Processing

The middlebox MUST send a KeyShareInfo Discovery record to the client if the client has indicated support for this extension, and has not

included the middlebox hash in the extension. The discovery record is followed by a MIDDLEBOX_PRESENT alert, breaking the connection. Similarly, if the hash is missing from the server's tls_keyshare extension, then the middlebox injects a KeyShareInfo Discovery record into the client-to-server stream. The server will usually then send an Alert record.
If the ServerHello specifies a ciphersuite that the middlebox does not support, or if it includes a TLS extension that might prevent the middlebox from processing, then the middlebox MAY send a KeyShareInfo Reject record with all unacceptable ciphersuites and extension numbers, followed by a MIDDLEBOX_PRESENT alert.

4. Middlebox Discovery

Discovering that the middlebox is present has already been described in Section 3.1. The client that is not aware of the presence of the middlebox receives a KeyShareInfo Discovery record followed by a MIDDLEBOX_PRESENT alert message.
Discovering that a middlebox in no longer on the path is trickier, because the superfluous KeyShareInfo Keys records do not lead to any observable effects for the client. We suggest that the client keep a list of discovered middleboxes, and periodically clear entries from the list, requiring a repeated discovery. System events such as a change to host IP address, a reboot or the computer entering sleep mode MAY be used as triggers for clearing the list.

5. Security Considerations

To be added

6. IANA Considerations

To be added.

7. References

7.1. Normative References

[1]	Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
[2]	Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J. and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 4366, April 2006.
[3]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

7.2. Informative References

[1]	Rescorla, E, "HTTP Over TLS", RFC 2818, May 2000.

[2]	McGrew, D and P Gladstone, "TLS Proxy Server Extension", Internet-Draft draft-mcgrew-tls-proxy-server-00, July 2011.

Author's Address

Yoav Nir Nir Check Point Software Technologies Ltd. 5 Hasolelim st. Tel Aviv, 67897 Israel EMail: ynir@checkpoint.com