Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 3, 2018

R. Housley
Vigil Security
R. Droms
Google

March 2, 2018

TLS 1.3 Option for Negotiation of Visibility in the Datacenter
draft-rhrd-tls-tls13-visibility-01

Abstract

Current drafts of TLS 1.3 do not include the use of the RSA handshake. While (EC) Diffie-Hellman is in nearly all ways an improvement over the TLS RSA handshake, the use of (EC)DH has impacts certain enterprise network operational requirements. The TLS Visibility Extension addresses one of the impacts of (EC)DH through an opt-in mechanism that allows a TLS client and server to explicitly grant access to the TLS session plaintext.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 3, 2018.

Copyright Notice

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

1. Introduction

Unlike earlier versions of TLS, current drafts of TLS 1.3 [I-D.ietf-tls-tls13] do not provide support for the RSA handshake -- and have instead adopted ephemeral-mode Diffie-Hellman (DHE) and elliptic-curve Diffie-Hellman (ECDHE) as the primary cryptographic key exchange mechanism used in TLS.

While ephemeral (EC) Diffie-Hellman is in nearly all ways an improvement over the TLS RSA handshake, the use of these mechanisms has impacts on certain enterprise operational requirements. Specifically, the use of ephemeral ciphersuites prevents the use of current enterprise network monitoring tools such as Intrusion Detection Systems (IDS) and application monitoring systems, which leverage the current TLS RSA handshake to passively decrypt and monitor intranet TLS connections made between endpoints under the enterprise's control. This traffic includes TLS connections made from enterprise network security devices (firewalls) and load balancers at the edge of the enterprise network to internal enterprise TLS servers. It does not include TLS connections traveling over the external Internet.

Such monitoring of the enterprise network is ubiquitous and indispensable in some industries, and is required for effective and safe operation of their enterprise networks. Loss of this capability may slow adoption of TLS 1.3 or force enterprises to continue to use outdated and potentially vulnerable technology.

The TLS Visibility Extension provides an option to enable visibility into a TLS 1.3 session by an authorized third party. Use of the extension requires opt-in by the TLS client when it initiates a TLS 1.3 session. The TLS server then opts-in by including keying material that will enable decryption in the TLS Visibility Extension. The presence of the TLS Visibility Extension provides a clear indication that other parties have been granted access to the TLS session plaintext. The keying material in the TLS Visibility Extension is encrypted and can only be decrypted by authorized parties that have been given the private key from a managed Diffie-Hellman key pair.

2. Terminology

Two key pairs are used with the TLS Visibility Extension for encryption of the session secrets:

SSWrapDH1: generated externally and the public key is provided to the TLS 1.3 server prior to use of the TLS Visibility Extension; the corresponding private key is provided to the parties that are authorized to access the TLS session plaintext.

SSWrapDH2: an ephemeral key pair that is generated by the TLS 1.3 server for each TLS 1.3 session that uses the TLS Visibility Extension; the server keeps the private key confidential, and passes the public key to the other parties in the TLS Visibility session.

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].

3. Extension Overview

Prior to the use of the TLS Visibility Extension, the SSWrapDH1 key pair is generated, possibly by an enterprise key manager. The private key is passed to the parties that are authorized to access the TLS session plaintext. The server is provisioned with the public key. When a new TLS 1.3 session is initiated, the client includes an empty TLS Visibility Extension in the ClientHello. The server then generates a SSWrapDH2 ephemeral key pair. The server will then:

- Generate a key, Ke, from the SSWrapDH1 public key and the SSWrapDH2 private key.

- Encrypt the TLS 1.3 session Early Secret (if one exists) and Handshake Secret (session secret) using Ke.

- Send an identifier for the SSWrapDH1 public key (called the fingerprint), the SSWrapDH2 public key, and the encrypted session secrets in the TLS Visibility Extension in the ServerHello message.

To decrypt the TLS 1.3 session, a party that is authorized to access the TLS session plaintext must be given the SSWrapDH1 private key. The party then:

- Obtains the SSWrapDH1 public key from the TLS Visibility extension

- Uses the SSHWrapDH1 private key and the SSWrapDH2 public key to generate Ke

- Uses Ke to decrypt the session secrets carried in the TLS Visibility extension

- Uses the session secrets to derive the keying material needed decrypt the TLS 1.3 session

4. TLS Visibility Extension

This section specifies the "tls_visibility" extension, which is carried in the ClientHello message and the ServerHello message.

The general extension mechanisms enable clients and servers to negotiate the use of specific extensions. As specified in [I-D.ietf-tls-tls13], clients request extended functionality from servers with the extensions field in the ClientHello message. If the server responds HelloRetryRequest, then the client sends another ClientHello message that includes the same extensions field as the original ClientHello message.

Most server extensions are carried in the EncryptedExtensions message; however, the "tls_visibility" extension is carried in the ServerHello message in a manner similar to the "key_share" and "pre_shared_key" extensions. It is only present in the ServerHello message if the server wants to enable TLS Visibility for some other parties and the client has offered the "tls_visibility" extension in the ClientHello message.

The "tls_visibility" extension MAY appear in the CH (ClientHello message) and SH (ServerHello message). It MUST NOT appear in any other messages. The "tls_visibility" extension MUST NOT appear in the ServerHello message unless "tls_visibility" extension appeared in the preceding ClientHello message. If an implementation recognizes the "tls_visibility" extension and receives it in any other message, then the implementation MUST abort the handshake with an "illegal_parameter" alert.

The Extension structure is defined in [I-D.ietf-tls-tls13]; it is repeated here for convenience.

```
struct {
    ExtensionType extension_type;
    opaque extension_data<0..2^16-1>;
} Extension;
```


The "extension_type" identifies the particular extension type, and the "extension_data" contains information specific to the particular extension type.

This document specifies the "tls_visibility" extension type, adding one new type to ExtensionType:

```
enum {
    tls_visibility(TBD), (65535)
} ExtensionType;
```

The "tls_visibility" extension is relevant when the client and server choose to enable one or more other parties to decrypt the TLS session.

Clients MUST include the "tls_visibility" extension in the ClientHello message to indicate their willingness for other parties to decrypt the TLS session. The server responds with data that enables the other parties to derive the keying material needed to decrypt the session if they are in possession of the indicated ECDH

private key.

```
  struct {
      select (Handshake.msg_type) {
          case client_hello:  Empty;
          case server_hello:  WrappedSessionSecrets visibility_data;
      };
  } TLSVisibilityExtension;

  struct {
      opaque early_secret<1..255>;
      opaque hs_secret<1..255>;
  } SessionSecrets;

  struct {
      opaque fingerprint<20>;
      opaque key_exchange<1..2^16-1>;
      opaque wrapped_secrets<1..2^16-1>;
  } WrappedSessionSecrets;
```

The fields in WrappedSessionSecrets are used as follows:

o "fingerprint" contains the leftmost 20 octets of the SHA-256 hash of SSWrapDH1 public key that was used by the server to compute the

session secret wrapping key. The public key is DER-encoded in the SubjectPublicKeyInfo [RFC5280] for the SHA-256 hash computation. The key manager tells the server which AEAD algorithm to use with this SSWrapDH1 public key at the time it is distributed.

o "key_exchange" contains the SSWrapDH2 ephemeral public key generated by the server on the same elliptic curve as the SSWrapDH1 public key identified by the "fingerprint". The server uses the SSWrapDH2 ephemeral private key and the SSWrapDH1 public key identified by the "fingerprint" to compute a shared secret value, called Z, and then uses HKDF [RFC5869] to produce the session secret wrapping key, called Ke, and an AEAD nonce, if one is needed by the AEAD algorithm [RFC5116]. The details of the key agreement process are described in Section 5.

- "wrapped_secrets" contains the SessionSecrets structure encrypted with the AEAD algorithm under Ke. The details of the encryption process are described in Section 5.

The fields in SessionSecrets are used as follows:

- "early_secret" contains the Early Secret that was derived from the pre-shared key. If this session did not use a pre-shared key, then the Early Secret is HKDF-Extract(0, 0).

- "hs_secret" contains the handshake key that was computed using (EC)DHE.

5. Session Secret Wrapping

The input to the encryption process is the encoded SessionSecrets structure, and the ciphertext is carried in the "wrapped_secrets" field in the WrappedSessionSecrets structure. The session secret wrapping key, called Ke, and an AEAD nonce, if one is needed by the AEAD algorithm [RFC5116] are used to perform the encryption. For example, AES-KEY-WRAP-256 [RFC5649] does not require a nonce, but AES-GCM-128 [GCM] does require a nonce.

The "key_exchange" field of the WrappedSessionSecrets structure contains the SSWrapDH2 ephemeral public key generated by the server on the same elliptic curve as the SSWrapDH1 public key identified by the "fingerprint" field of the WrappedSessionSecrets structure. The server uses the SSWrapDH2 ephemeral private key and the SSWrapDH1 public key to compute a shared secret value, called Z, and then uses HKDF [RFC5869] to produce the Ke and the nonce:


```
PRK = HKDF-Extract(0x00, Z)
Ke = HKDF-Expand(PRK, "tls_vis_key", AEAD_key_size)
nonce = HKDF-Expand(PRK, "tls_vis_nonce", AEAD_nonce_size)
```

The length of the ciphertext can be longer than the input plaintext, depending on the AEAD algorithm that is used. The AEAD algorithm is distributed to the server along with the SSWrapDH1 public key, so

there is no need to carry an explicit algorithm identifier.

Encryption is performed as follows:

```
wrapped_secrets = AEAD-Encrypt(Ke, nonce, SessionSecrets)
```

Other parties use the SSWrapDH2 ephemeral public key from the "key_exchange" field of the WrappedSessionSecrets structure and the SSWrapDH1 private key that is associated with the "fingerprint" field of the WrappedSessionSecrets structure to compute a shared secret value, called Z. The SSWrapDH1 private key and the AEAD algorithm are obtained in advance. Then, Z is used to produce the Ke and the nonce as specified above. To unwrap the session secrets, decryption is performed as follows:

```
SessionSecrets = AEAD-Encrypt(Ke, nonce, wrapped_secrets)
```

The result is either the plaintext of the SessionSecrets structure or an error indicating that the decryption failed. An integrity check is performed as part of the decrypt operation.

6. Alternative Approaches

This section captures the rationale for pursuing this approach to TLS visibility instead of the various alternative approaches.

Server uses a static Diffie-Hellman key pair: Instead of generating ephemeral Diffie-Hellman key pairs, the server reuses a static Diffie-Hellman key pair. The static private Diffie-Hellman key gets shared with the points that need visibility. While this approach scales, the TLS client is unaware of the sharing. In addition, this enables visibility of data of all clients communicating with the server, versus only those that opt-in to visibility.

Export of ephemeral keys: In large enterprises there will be

billions of ephemeral keys to export and distribute. Transporting these keys to tools for decryption of packets in real time will be difficult, adding greatly to the complexity of the solution.

Export of decrypted traffic from TLS proxy devices: Decrypting traffic only at the edge of the enterprise datacenter does not meet all of the enterprise requirements, which include troubleshooting, fraud detection, and network security monitoring. Further, the number of TLS proxies needed are quite costly, add latency, and increase production risk.

Continue to use TLS 1.2 within the enterprise network: TLS 1.2 could be used within the enterprise network (with TLS 1.3 outside) to enable TLS visibility via RSA key transport. However, TLS 1.3 has security improvements over TLS 1.2. At some point in the future, TLS 1.2 will not longer be supported and available in enterprise applications and protocol implementations. In addition, based on experience, standards bodies will deprecate the use of TLS 1.2 and require enterprise networks to move to TLS 1.3.

Reliance on TCP/IP headers: TCP and IP headers are not adequate for enterprise requirements. Troubleshooting, fraud detection, and network security monitoring need access to the plaintext payload. For example, troubleshooters must be able to find specific transactions, user identifiers, session identifiers, URLs, and time stamps.

Reliance on application and server logs: Logging is not adequate for enterprise requirements. Code developers cannot anticipate every possible problem for logging, and system administrators turn much of the logging off to conserve system resources.

Troubleshooting and malware analysis at the endpoint: Endpoints are focused on providing a service, and they cannot handle the additional burden of the various enterprise monitoring requirements.

Adding TCP/UDP extensions: An important part of troubleshooting, network security monitoring, etc. is analysis of the application-specific payload of the packet. It is not possible to anticipate ahead of time, among thousands of unique applications, which fields in the application payload will be important.

7. IANA Considerations

IANA is requested to update the TLS ExtensionType Registry to include "tls_visibility" with a value of [TBD] and the list of messages "CH, SH" in which the "tls_visibility" extension may appear.

8. Security Considerations

The use of a TLS protocol extension ensures that both the TLS client and the TLS server are aware that other parties have visibility into the TLS session plaintext. However, the approach used here does not allow those parties to masquerade since they do not have the ability to sign the Finished message in the TLS handshake.

Use of the TLS Visibility extension represents a deliberate introduction by the client and server of other parties that can access the TLS session plaintext. Deployments that choose to make use of this extension should carefully consider the risks associated with the change to the Forward Secrecy. In particular, Forward Secrecy will not begin for sessions where the TLS Visibility Extension is used until all of these events take place:

(1) The server has securely discarded the session secrets.

(2) The server has securely discarded the session secret wrapping key.

(3) The client has securely discarded the session secrets.

(4) The other parties have securely discarded the session secrets.

(5) The other parties have securely discarded the session secret wrapping key.

(6) The other parties have securely discarded the ECDHE private key that was used to derive the session secret wrapping key.

By agreeing to the use of the TLS Visibility extension, the client is aware that the TLS session plaintext will be accessible to any other party that has access to the ECDHE private key that was used to derive the session secret wrapping key. It is envisioned that the server and other parties will all be under a single administrative control; however, the TLS Visibility extension does not guarantee any particular scope for the distribution of the ECDHE private keys.

The SSWrapDH1 and SSWrapDH2 key size and parameters MUST be selected

to provide the same level (or more) of security as the (EC)DHE key used in the TLS Handshake. Similarly, the Sessions Secret Wrapping

key size and algorithm MUST be selected to provide the same level (or more) of security as the AEAD cipher used with the TLS Record protocol. If weaker key sizes, parameters or algorithms are used, the attacker will find it easier to obtain the session secrets from the TLS Visibility extension.

9. Acknowledgments

Matthew Green was the primary author of [I-D.green-tls-static-dh-in-tls13], which describes an earlier solution to the TLS 1.3 session visibility problem. Nick Sullivan and Richard Barnes suggested the use of client and server opt-in. Peter Wu suggested the use of HKDF-Expand to get a nonce. Nalini Elkins, Steven Fenter, Sinok Lao, Andrew Kennedy, Darin Pettis, Tim Polk, Andrew Regenscheid, Murugiah Souppaya, and Paul Turner contributed through discussion to the development of this document.

10. References

10.1. Normative References

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-24 (work in progress), February 2018.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869,

DOI 10.17487/RFC5869, May 2010,
<https://www.rfc-editor.org/info/rfc5869>.

10.2. Informative References

[GCM] Dworkin, M., "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC", NIST Special Publication 800-38D, November 2007.

<http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf>

[I-D.green-tls-static-dh-in-tls13]
Green, M., Droms, R., Housley, R., Turner, P., and S. Fenter, "Data Center use of Static Diffie-Hellman in TLS 1.3", draft-green-tls-static-dh-in-tls13-01 (work in progress), July 2017.

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <https://www.rfc-editor.org/info/rfc5116>.

[RFC5649] Housley, R. and M. Dworkin, "Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm", RFC 5649, DOI 10.17487/RFC5649, September 2009, <https://www.rfc-editor.org/info/rfc5649>.

Authors' Addresses

Russ Housley
Vigil Security, LLC
918 Spring Knoll Drive
Herndon, VA 20170
USA

Email: housley@vigilsec.com

Ralph Droms
Google
355 Main Street

Cambridge, MA 02142
USA

Email: rdroms.ietf@gmail.com