LURK
Internet-Draft
Intended status: Standards Track
Expires: January 3, 2019

D. Migault
Ericsson

July 02, 2018

LURK Extension version 1 for (D)TLS 1.3 Authentication
draft-mglt-lurk-tls13-00

Abstract

This document describes the LURK Extension 'tls13' which enables interactions between a LURK Client and a LURK Server in a context of authentication with (D)TLS 1.3.

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 January 3, 2019.

Table of Contents

1. Introduction

This document defines a LURK extension for TLS 1.3. This document assume s the reader is familiar with [I-D.mglt-lurk-lurk] that describes the LURK architecture as well as the LURK Protocol and the integration of the LURK extensions. The motivations for the LURK Extension TLS 1.3 are similar to those for the LURK extension of TLS 1.2 [I-D.mglt-lurk-tls12].

LURK defines an interface to a Cryptographic Service that stores the security credentials - Typically the PSKs and private keys. Interactions with the Cryptographic Service can be performed by the TLS Client as well as by the TLS Server.

The TLS Server expects from the Cryptographic Service:

- To retrieve the necessary keys to complete the handshake. This typically includes the [sender]_handshake_traffic_secret to generate the keys necessary to encrypt the handshake extensions and messages, the [sender]_application_traffic_secret_N keys to protect the application data or the exporter_master_secret when needed.

- To generate of Handshake message or extensions that authenticate the TLS Server. This typically includes the CertificateVerify and Finished message.

- To retrieve NewSessionTicket to enable the TLS Client to perform session resumption.

The TLS Client expetcs from the Cryptographic Service:

- To retrieve the necessary keys to complete the handshake.
- To "provision" resumption_master_secret, so the PSK can be used when the TLS Client is using session resumption using a NewSession ticket.
- To generate of Handshake message or extensions that authenticate the TLS Client. This typically includes the CertificateVerify and Finished message.

Note that the TLS Server MAY interact with a single exchange with the Cryptographic Service, the TLS Client is expected to retrieve the [sender]_handshake_traffic_secret to generate the keys encrypt the handshake extensions and messages to decrypt the messages/extensions received from the TLS Server, prior to request for the generation of the CertificateVerify or Finished message.

2. LURK Header

LURK / TLS 1.3 is a LURK Extension that introduces a new designation "tls13". This document assumes that Extension is defined with designation set to "tls13" and version set to 1. The LURK Extension extends the LURKHeader structure defined in [I-D.mglt-lurk-lurk] as follows:

```
enum {
    tls13 (2), (255)
} Designation;

enum {
   capabilities (0), ping (1), rsa_master (2),
   rsa_extended_master (3), ecdhe (4), (255)
}TLS13Type;


enum {
    // generic values reserved or aligned with the
    // LURK Protocol
    request (0), success (1), undefined_error (2),
    invalid_payload_format (3),

    //code points for ecdhe authentication
    invalid_ec_type (9), invalid_ec_curve (10),
    invalid_poo_prf (11), invalid_poo (12), (255)
}TLS13Status

struct {
     Designation designation = "tls12";
     int8 version = 1;
} Extension;

struct {
    Extension extension;
    select( Extension ){
        case ("tls13", 1):
            TLS12Type;
    } type;
    select( Extension ){
        case ("tls13", 1):
            TLS13Status;
    } status;
    uint64 id;
    unint32 length;
} LURKHeader;
```

3. handshake_server_key

This exchange is only expected to be performed by a TLS Client. The server_handshake_key is necessary for the TLS Client to decrypt the handshake message/extensions encrypted by the TLS Server.

Interaction with a Cryptographic Service MAY be required when the PSK is protected by the Cryptographic Service.

3.1. Request Payload

```
enum { sha256 (0), (255) } TranscriptHash;

enum { psk_raw (0), psk_ticket (1), (255) } PSKType

struct {
    PSKType psk_type;
    select( psk_type ){
        case raw_psk :
            opaque raw_psk<0..2^16-1>;
        case identity_psk :
            OfferedPsks offered_psk  // {{I-D.ietf-tls-tls13}} section 4.2.11
} PSK

struct {
    select ( ke_mode ){  // {{I-D.ietf-tls-tls13}} section 4.2.9
        case psk_ke :
            PSK psk
        case psk_dhe_ke :
            PSK psk
            NamedGroup dh_group; // {{I-D.ietf-tls-tls13}} section 4.2.7
            opaque dhe_secret<1..2^16-1>;
    }
} KeyScheduleInputSecrets


enum {
    sha256 (0) (255)
} PFSAlgorithm

struct {
    PFSAlgorithm pfs;  // {{I-D.mglt-lurk-tls12}} section 4.1
    TranscriptHash h;
    PskKeyExchangeMode ke_mode // {{I-D.ietf-tls-tls13}} section 4.2.9
    opaque handshake_context<0..2^32-1>
    KeyScheduleInputSecrets secrets
} HandshakeServerKeyRequest
```

psk_type indicates how the PSK is provisioned to initiate the key schedule as described in [I-D.ietf-tls-tls13] section 7.1. The type psk_raw indicates the PSK is explicitly provided. The type psk_ticket indicates the PSK is generated from the ticket as described in [I-D.ietf-tls-tls13] section 4.6.1.

pfs the one-way hash function (OWHF) used by LURK to implement Perfect Forward Secrecy.

h the hash function used by the Transcript-Hash [I-D.ietf-tls-tls13] section 4.4.1.

offered_psk reuses the OfferedPsks described in [I-D.ietf-tls-tls13] section 4.2.11. The PSK structure only allow a single PSK, thus OfferedPsks MUST represent a single PSK.

ke_mode defines pre shared key exchange defined in [I-D.ietf-tls-tls13] section 4.2.9. It indicates whether the key exchange considers a (EC)DHE key establishment or not in addition to the PSK.

dh_group reuses the structure NamedGroup of [I-D.ietf-tls-tls13] section 4.2.7 to indicate the curve or the group used in (EC)DHE key establishment.

handshake_context the necessary handshake context to generate the key as described in [I-D.ietf-tls-tls13] section 7.1. The handshake_context MUST be ClientHello...ServerHello.

secrets the necessary secret inputs (PSK, (EC)DHE) secret necessary for the key schedule of [I-D.ietf-tls-tls13] section 7.1.

3.2. Response Payload

```
struct {
    opaque server_handshake_key<0..2^32-1>
} HandshakeServerKeyResponse
```

server_handshake_key the server_handshake_key

3.3. LURK Client Behavior

The TLS Client establishing a TLS session with a TLS Server receives from the TLS Server a ServerHello message with additional encrypted messages such as the EncryptedExtensions, the Finished as well as the optional Certificate, CertificateVerify and Application Data message. The TLS Client needs to retrieve the server_handshake_key in order to decrypt these messages.

With ServerHello as the input message, the LURK Client initiates the exchange as described below:

Perfect Forward Secrecy Setting:

o Perfect Forward Secrecy is performed as described in [I-D.mglt-lurk-tls12] section 4.1.1 over the client_random. There is no gmt_unix_time as such ServerHello.random is generated as

follows: ~~~ ClientHello.random = pfs(client_random + "tls13_c pfs"); ~~~

Transcript Hash Setting:

- the value for transcript hash is provided by the configuration.

PSK Key Exchange Mode Setting:

- if the input message does not contains any key_share extension, the LURK Client sets ke_mode to psk_ke.

- if a key_share extension is present the LURK Client sets ke_mode to psk_dhe_ke.

Key Schedule Input Secret Setting:

- if the input message has no pre_shared_key extension, the LURK Client sets psk_type to psk_raw with a psk of length 0.

- if the input message has a pre_shared_key extension, the LURK Client provides the PSK that is not a PSK of zero length, as follows: ..* the PSK is provided explicitly by using the psk_type set to psk_raw with the explicit value of the PSK. This alternative is NOT RECOMMENDED, as it means the PSK is not known by the Cryptographic Service and is known outside this service. It may happen when the TLS Client is configured with a PSK while the Cryptographic Service is not provisioned with that PSK. The case where the PSK is provided for a session resumption is outside the scope of this document as the session_resumption_secret is never shared outside the Cryptographic Service. ..* the PSK is provided via NewSessionTicket. Upon receiving a selected_identity in the pre_shared_key extension, the LURK Client selects the corresponding local_ticket previously provided by the LURK Server during the previous handshake. local_ticket are internal structure used by LURK detailled in Section 4.4.1

3.4. LURK Server Behavior

Upon receiving a handshake_server_key request, the LURK server proceeds as follows:

Perfect Forward Secrecy Check:

- if pfs is not supported, an invalid_pfs erroro is returned.

- ClientHello.random is generated as described in Section 3.3 and the value is provided in handshake_context.

Transcript-Hash Check:

- if h is not a supported transcript-hash function and invalid_transcript_hash error is returned.

Handshake Check:

- if handshake does not contains a ClientHello...ServerHello an invalid_handshake error is returned.

PSKExchangeMode Check:

- if ke_mode is not supported an invalid_ke_mode error is returned.

KeyScheduleInputSecret Check check the validity of the secrets as well as the coherence wit the pre shared key exchange. These checking operations are subdivided into (EC)DHE Check and PSK Check operations:

(EC)DHE Check:

- if ke_mode is set to psk_dhe_ke and secret does not contain a (EC)DHE secret an invalid_secret error is returned.

- if the (EC)DHE secret does not match the expected length or the curve is not supported an invalid_ecdhe_secret error is returned.

PSK Check:

- if the psk type is not supported a invalid_psk_type is returned.

- if psk_type is psk_raw and the format of the psk is unexpected an invalid_psk_format error is returned.

- if the psk_type is psk_ticket:

..* if the number of psk or associated binder is more than 1, an invalid_ticket_format error is returned ..* there is no corresponding identity, an invalid_psk_ticket error is returned. * if the psk_type is identity_psk binder_key is generated as described in [I-D.ietf-tls-tls13] section 7.1. ..* if the binder associated to the psk does not match the one provided in the offered_psk and invalid_binder error is returned. The binder is computed as described in [I-D.ietf-tls-tls13] section 4.2.11.2. with the binder_key generated as described in [I-D.ietf-tls-tls13] section 7.1.

Key Generation:

- server_handshake_key is generated as described in [I-D.ietf-tls-tls13] section 7.1 and returned to the LURK Client.

4. auth

This exchange provides interactions with a Cryptographic Service both on the TLS Client side as well as the TLS Server side.

4.1. Request Payload

```
enum { server (0), client (1), post-handshake (2) } HandshakeMode;

struct {
    PFSAlgorithm pfs;  // {{I-D.mglt-lurk-tls12}} section 4.1
    TranscriptHash h;  // c, f
    PskKeyExchangeMode ke_mode // {{I-D.ietf-tls-tls13}} section 4.2.9
    select( ke_mode ){
        case : psk_dhe_ke
            Certificate certificate        // {{I-D.ietf-tls-tls13}} section
4.4.2
            SignatureScheme algorithm     // {{I-D.ietf-tls-tls13}} section
4.2.3.
    }
    HandshakeMode handshake_mode        // c, f
    opaque handshake_context<0..2^32-1> // c, f
    KeyScheduleInputSecrets secrets // f
    uint8 key_request
    uint8 ticket_number
} AuthRequest

c: structure used for the CertificateVerify message
f: structure used for the Finished message
```

pfs, h, ke_mode, handshake_context and secrets are define in Section 3.1

certificate end point certificate defined in [I-D.ietf-tls-tls13] section 4.4.2.

algorithm signature algorithm used defined in [I-D.ietf-tls-tls13] section 4.2.3.

handshake_mode defines the specific Handshake Context and Base Key necessary to compute authentication messages as defined in [I-D.ietf-tls-tls13] section 4.4. The handshake_mode set to server indicates the LURK exchange is performed by the TLS Server while the handshake_mode set to client or post-handshake indicates the LURK exchange is performed by the TLS Client.

handshake_context Handshake Context has defined in [I-D.ietf-tls-tls13] section 4.4.

key_request indicates optional requested keys. The bit is set to 1 to indicate the key is being requested by the LURK Client. ..* bit 0 : client_handshake_traffic_secret ..* bit 1 : server_handshake_traffic_secret ..* bit 2 : client_application_traffic_secret_0 ..* bit 3 : server_application_traffic_secret_0 ..* bit 4 : exporter_master_secret ..* bit 5-7: set to 0

ticket_number indicates the expected number of session resumption tickets. When requested by the TLS Client the ticket_number is expected to be 0 or 1. When requested by the TLS Server the number can be larger.

4.2. Response Payload

```
struct{
    opaque key<0..2^16-1>
} Key

struct {
    uint8 key_index
    opaque key_list<0..2^32-1>
} Keys

struct {
    Keys keys
    CertificateVerify certificate_verify
    Finished finished
    NewSessionTicket ticket_list<0..2^32-1>
} AuthResponse
```

key_index follows the same syntax as key_request in Section 4.1.

key_list :the list of keys indicated by key_index.

ticket_list list of NewTicketSessions

4.3. LURK Client Behavior (TLS Server)

On a TLS Server, the LURK Server initiates the LURK exchange after receiving the ClientHello from the TLS Client. The purpose of this exchange is to retrieve the CertificateVerify, Finished, and the necessary keys to:

- encrypt the EncryptedExtensions, Finished and optional CertificateRequest Certificate and CertificateVerify message: server_handshake_traffic_secret

- encrypt the optional Application Data message: server__application_traffic_secret_N

- decrypt the future Finished or optional Certificate and CertificateVerify message sent by the TLS Client: client_handshake_traffic_secret.

- decrypt the future Application Data message with the client__application_traffic_secret_N

Perfect Forward Secrecy Setting:

- Perfect Forward Secrecy is performed as described in [I-D.mglt-lurk-tls12] section 4.1.1 over the server_random. There is no gmt_unix_time as such ServerHello.random is generated as follows: ~~~ ServerHello.random = pfs(server_random + "tls13_s pfs"); ~~~

The LURK Client proceeds to the Transcript Hash Setting PSK Key Exchange Mode Setting and the Key Schedule Input Secret Setting as described in Section 3.3.

Handshake Mode Setting:

- If the LURK Client sets the handshake_mode to "server".

Handshake Setting: The handshake is set as described in [I-D.ietf-tls-tls13] section 4.4.

Key Request Setting:

- key_request MUST have the Bit 0 and Bit 1 set to retrieve the [sender]_handshake_traffic_secret.

- key_request MUST have the Bit 2 and Bit 3 set to retrieve the [sender]_application_traffic_secret_N

- Key_request MAY have Bit 4 set if there is a need to use the extractor.

Upon receiving the AuthResponse, the TLS Server encrypts the messages and pursue the TLS handshake as defined in [I-D.ietf-tls-tls13].

4.4. LURK Client Behavior (TLS Client)

On a TLS Client the LURK Client initiates an AuthRequest in order to compute the Finished and optional CertificateVerify as well as to retrieve the necessary keys to:

- encrypt the Finished and optional Certificate and CertificateVerify message: client_handshake_traffic_secret
- encrypt the optional Application Data message: server__application_traffic_secret_N
- decrypt the future Application Data message with the server__application_traffic_secret_N

The TLS Client has decrypted the encrypted handshake messages sent by the TLS Server by retrieving the server_handshake_traffic_secret with an HandshakeServerKeyRequest.

The LURK Client proceeds to Perfect Forward Secrecy Setting, Transcript Hash Setting, Key Schedule Input Secret Setting as described in Section 3.3.

If the TLS Client has received a CertificateRequest from the TLS Server, the LURK Client:

- sets the ke_mode to psk_dhe_ke. Note that the value is not correlated to the value agreed psk_key_exchange_modes between the TLS Client and the TLS Server. Instead it indicates the necessity to generate a CertificateVerify.
- provides the Certificate associated to the private key of the TLS

Handshake Mode Setting:

- If the LURK Client is initiating a LURK exchange on behalf of a TLS Client it sets the handshake_mode to: ..* "client" when the LURK exchange occurs during the TLS handshake. ..* "post-authentication" when the LURK exchange occurs outside the TLS handshake.

Handshake Setting:

- set handshake_context as defined in [I-D.ietf-tls-tls13] section 4.4.

Key Request Setting:

- key_request SHOULD have the Bit 0 unset, as client_handshake_traffic_secret is already known by the TLS Client
- key_request MUST have the Bit 1 set to retrieve the server_handshake_traffic_secret.
- key_request MUST have the Bit 2 and Bit 3 set to retrieve the [sender]_application_traffic_secret_N
- Key_request MAY have Bit 4 set if there is a need to use the extractor.

Ticket Number Setting:

- If the TLS Client want to performed further session resumption, ticket_number is set to 1 and 0 otherwise.

Upon receiving the AuthResponse, the LURK Client has the necessary information to proceed the TLS handshake. The ticket_list is a list of local_ticket. The list MUST have a maximum of one local_ticket. The LURK Client is expected to manage the local_tickets as described in Section 4.4.1

4.4.1. Local Ticket

local_ticket re-uses the NewSessionTicket structure in two different ways depending if the LURK exchange is initiated by a TLS Client or by a TLS Server.

- ticket provided to the TLS Server (by the LURK Server) are new_session_ticket, expected to be forwarded to the TLS Client.
- tickets provided to the TLS Client (by the LURK Server) are local_ticket. These local_tickets are only expected to be used between the LURK Client and the LURK Server of the TLS Client.

During the initial handshake, the TLS Client has received a local_ticket from the LURK Server and a new_session_ticket from the TLS Server. The TLS Client updates the local_ticket as follows: the ticket_nonce and extensions fields of the new_session_ticket are copied to the local_ticket.

When the TLS Server provides more than one new_session_ticket tickets, these tickets are expected to have different nonce. On the other hand a single local_ticket will be provided by the LURK Server. The TLS Client generates an associated local_ticket for each new_session_ticket. All of them are generated from the local_ticket provided by the LURK Server.

Though the new_Session_ticket and the local_ticket have different meanings, a TLS Client will not be able to perform session resumption without the corresponding local_ticket. More specifically, the TLS Client MUST:

- remove local_tickets and new_session_tickets that have expired
- remove local_tickets that have no associated new_session_tickets
- remove new_session_tickets that have no associated local_tickets In all these cases, a new handshake will be renegotiated. Note that this gives the Cryptographic Service the ability to define the maximum time a new_session_ticket can be used.

4.5. LURK Server Behavior

Upon receiving a handshake_server_key request, the LURK server proceeds as follows:

Perfect Forward Secrecy Check is performed as Section 3.3 using the pfs, and client_random (resp. server_random) as described in Section 3.3 (resp. Section 4.3).

Transcript-Hash Check, PSKExchangeMode Check, KeyScheduleInputSecrets Check are performed as described in Section 3.4

HandshakeMode Check:

- if the mode is not supported a invalid_handshake_mode error is returned. This typically prevents a TLS Client to perform computation expected to happen on the TLS Server, or to distinguish and authorize client authentication performed during the handshake or post handshake.

Handshake Check:

- if the hanshake_context does not match the expected handshake context as defined in [I-D.ietf-tls-tls13] section 4.4. an invalid_handshake error is returned.

CertificateVerify Check:

- if ke_mode is set to psk_dhe_ke and the certificate is not supported an invalid_certificate error is returned
- if ke_mode is set to psk_dhe_ke and the algorithm is not supported an invalid_signature_scheme error is returned

Keys are generated as described in [I-D.ietf-tls-tls13] section 7.1. This includes the Base Key use to generate the Finished messages as well as the resumption_master_secret.

key_request is indicative and is used by the LURK Client to indicate the keys that are not necessarily needed in order to save bandwidth. The LURK Server SHOULD NOT responds with keys whose key_request bit is unset.

The CertificateVerify message is generated as described in [I-D.ietf-tls-tls13] section 4.4.3.

The Finished message is generated as described in [I-D.ietf-tls-tls13] section 4.4.4.

ticket_number indicates the number of NewSessionTicket. ticket_session have different meaning when used by the TLS Client or the TLS Server. When the LURK exchange is initiated by the LURK Client, the ticket_sessions are local_ticket and are only expected to be used between the LURK Client and the LURK Server. Such local_ticket avoids a direct communication of the resumption_master_secret. local_ticket follows the definition of new_session_tickets described in [I-D.ietf-tls-tls13] section 4.6.1. The LURK Server MUST have a zero length ticket_nonce and zero length extensions

When the LURK exchange is initiated by the TLS Server the tickets are new_session_tickets as described in [I-D.ietf-tls-tls13] section 4.6.1. As a result:

- if handshake_mode is set to server, the LURK Server SHOULD respond with a list of new_session_tickets that is not greater than the number indicated by ticket_number. The number of ticket MAY be defined by the LURK Server policies.

- if handshake_mode is set to client or post-handshake the LURK Server SHOULD respond with a list of local_ticket that is not greater than the number indicated by ticket_number. The list MUST NOT exceed one local_ticket.

5. Security Considerations

6. IANA Considerations

7. Acknowledgments

8. Normative References

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-28 (work in progress), March 2018.

[I-D.mglt-lurk-lurk]
Migault, D., "LURK Protocol version 1", draft-mglt-lurk-lurk-00 (work in progress), February 2018.

[I-D.mglt-lurk-tls12]
Migault, D., "LURK Extension version 1 for (D)TLS 1.2 and (D)TLS 1.1 Authentication", draft-mglt-lurk-tls12-00 (work in progress), February 2018.

Author's Address

Daniel Migault
Ericsson
8275 Trans Canada Route
Saint Laurent, QC 4S 0B6
Canada

EMail: daniel.migault@ericsson.com