Abstract

This document describes the LURK Extension ‘tls12’ which enables interactions between a LURK Client and a LURK Server in a context of authentication with (D)TLS 1.1 and (D)TLS 1.2.

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 August 13, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must 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.
# Table of Contents

1. Requirements notation ................................................. 3  
2. Introduction ..................................................................... 3  
3. Terminology and Acronyms ................................................ 4  
4. LURK Header .................................................................... 4  
5. rsa_master ........................................................................ 6  
   5.1. Request Payload ......................................................... 6  
   5.1.1. PFS-PRFA .............................................................. 7  
   5.2. Response Payload ....................................................... 7  
   5.3. LURK Client Behavior .................................................. 8  
   5.4. LURK Server Behavior .................................................. 8  
6. rsa_extended_master .......................................................... 9  
   6.1. Request Payload ......................................................... 9  
   6.2. Response Payload ....................................................... 10  
   6.3. LURK Client Behavior ................................................... 10  
   6.4. LURK Server Behavior ................................................... 10  
7. ecdhe .............................................................................. 10  
   7.1. Request Payload ......................................................... 10  
   7.2. Response Payload ....................................................... 12  
   7.3. LURK Client Behavior ................................................... 12  
   7.4. LURK Server Behavior ................................................... 12  
8. capabilities ...................................................................... 13  
   8.1. Request Payload ......................................................... 13  
   8.2. Response Payload ....................................................... 13  
   8.3. LURK Client Behavior ................................................... 15  
   8.4. LURK Server Behavior ................................................... 15  
9. ping .............................................................................. 15  
   9.1. Request Payload ......................................................... 15  
   9.2. Response Payload ....................................................... 15  
   9.3. LURK Client Behavior ................................................... 15  
   9.4. LURK Server Behavior ................................................... 16  
10. Security Considerations .................................................... 16  
10.1. RSA ........................................................................... 16  
10.2. ECDHE ........................................................................ 17  
11. IANA Considerations ....................................................... 18  
12. Acknowledgments ............................................................ 19  
13. References ...................................................................... 20  
   13.1. Normative References .................................................. 20  
   13.2. Informative References ................................................ 20  
Appendix A. Example of LURK Exchanges ................................. 21  
   A.1. LURK/TLS RSA Master Secret ...................................... 22  
   A.2. LURK/TLS RSA Extended Master Secret ......................... 23  
   A.3. LURK/TLS ECDHE Signature ......................................... 25  
Author’s Address ................................................................. 26
1. Requirements notation

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. Introduction

This document describes the LURK Extension for TLS 1.1 and TLS 1.2 so the LURK Server can implement a Cryptographic Service in a TLS 1.1 [RFC4346], DTLS 1.1 [RFC4347], a TLS 1.2 [RFC5246] and DTLS 1.2 [RFC6347] context.

More specifically, the LURK Server will be in charge of performing the cryptographic operations associated to the private key of the TLS Server, while other aspects of the termination of the TLS session is handled by other services in the same administrative domain or in a different administrative domain. Most Cryptographic Operations are related to the TLS authentication and the current document limits the Cryptographic Operations to the following authentication methods: RSA and ECDHE_RSA defined in [RFC5246], [RFC6347] as well as ECDHE_ECDSA defined in [RFC4492].

A more detailed description of some use cases foreseen in a TLS context can be found in [I-D.mglt-lurk-tls-use-cases].

HTTPS delegation has been the main concern of the Content Delivery Networks Interconnection (cdni) Working Group and several mechanisms have been designed to delegate the load from an upstream entity to a downstream entity. Entities can be of different nature and may designated differently according to the context. Typically designations includes Content Owner, CDN Provider, Domain Name Owner for example. [I-D.fieau-cdni-https-delegation] provides a details comparison of the various mechanisms applies to the CDN Interconnection, and the remaining of this section positions these mechanisms at a very high level view.

STAR [I-D.ietf-acme-star], [I-D.sheffer-acme-star-request] describes a methods where the domain name owner or the content owner orchestrates the refreshing process between a CA and the CDN (terminating the TLS session). The CDN refreshes regularly and automatically its certificates using [I-D.ietf-acme-acme], which allows the use of short term certificates.

Delegated credentials [I-D.rescorla-tls-subcerts] consists having a certificate that enables the servers to generates some "delegated credentials".
STAR and "delegated credentials" both require some changes performed by the CA - new certificate type for the delegated credentials and new interfaces for the delegated and delegating entity for STAR. In both case the TLS Client authenticates the delegated entity. While STAR does not require changes on the TLS Client, the "delegated credential" solution does. In both cases, the delegation is controlled by limiting in time (7 days), which is also the limit of use of a stolen key or a rogue server. Such delegation provides a high scalability of the architecture and prevents additional delays when a TLS session is established.

The LURK Architecture [I-D.mglt-lurk-lurk] and the LURK Extension 'tls12' do not proceed to the delegation of the HTTPS delegation by delegating the entire TLS termination. Instead, the TLS termination is split into sub services, for example one associated to the networking part and one associated to the cryptographic operation. While micro services associated to the networking part are delegated, the micro service associated to the cryptographic operation may not be delegated. As a result, LURK Architecture is focused on the protection of the Cryptographic Material and prevents leakage of the Cryptographic Material for example by avoiding node exposed to the Internet to host the Cryptographic Material. In addition, LURK provides means to instantaneously suspend the delegation with a suspicious node. On the other hand the LURK Extension ‘tls12’ introduces some latency, and is not as scalable as STAR or delegated credential solutions.

The LURK Extension ‘tls12’ is seen as a complementary to the STAR and "delegated credentials". The LURK Extension ‘tls12’ is a back end solution that does not require any modifications from TLS Client or the CA. It is also aimed at protecting the Cryptographic Material.

LURK may also be deployed within an administrative domain in order to provide a more controled deployment of TLS Servers.

3. Terminology and Acronyms

This document re-uses the terminology defined in [I-D.mglt-lurk-lurk].

4. LURK Header

LURK / TLS 1.2 is a LURK Extension that introduces a new designation "tls12". This document assumes that Extension is defined with designation set to "tls12" and version set to 1. The LURK Extension extends the LURKHeader structure defined in [I-D.mglt-lurk-lurk] as follows:
enum {
    tls12 (1), (255)
} Designation;

enum {
    capabilities (0), ping (1), rsa_master (2),
    rsa_extended_master (3), ecdhe (4), (255)
} TLS12Type;

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

    // code points for rsa authentication
    invalid_key_id_type (4), invalid_key_id (5),
    invalid_tls_version (6), invalid_tls_random (7), invalid_prf (8),
    invalid_encrypted_premaster (9)

    // code points for ecdhe authentication
    invalid_ec_type (10), invalid_ec_basistype (11),
    invalid_ec_curve (12), invalid_ec_point_format (13),
    invalid_poo_prf (138), invalid_poo (139), (255)
} TLS12Status

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

struct {
    Extension extension;
    select (Extension ){
        case ("tls12", 1):
            TLS12Type;
    } type;
    select (Extension ){
        case ("tls12", 1):
            TLS12Status;
    } status;
    uint64 id;
    uint32 length;
} LURKHeader;
5. rsa_master

A exchange of type "rsa_master" enables the LURK Client to delegate the RSA Key Exchange and authentication as defined in [RFC5246]. The LURK Server returns the master secret.

5.1. Request Payload

A rsa_master request payload has the following structure:

```c
enum {
    sha256_32 (0), (255)
} KeyPairIdType;

struct {
    KeyPairIdType type;
    opaque data; // length defined by the type
} KeyPairID;

enum{
    sha256_null (0), sha256_sha256 (1) (255)
} PRFAlgorithm

struct {
    KeyPairID key_id;
    Random client_random; // see RFC5246 section 7.4.1.2
    Random server_random;
    ProtocolVersion tls_version; // see RFC5246 section 6.2.1
    PRFAlgorithm prf; // see RFC5246 section 6.1
} TLS12Base;

struct {
    TLS12Base base;
    EncryptedPreMasterSecret pre_master;
    // see RFC5246 section 7.4.7.1
    // Length depends on the key.
} TLS12RSAMasterRequestPayload;

key_id  The identifier of the RSA key. This document defines sha256_32 format which takes the 32 first bits of the hash of the public key using sha256.

prf  the set of pseudo random functions used. This document considers two distinct Pseudo Random Function Algorithm (PRFA):

The Master Secret PRFA (MS-PRFA) used to generate the master secret as defined in [RFC5246] Section 6.1 and the Perfect Forward Secrecy PRFA (PFS-PRFA) used to implement PFS. This representation used in this document is MS-PRFA_PFS-PRFA.
client_random  The random value associated to the TLS Client as defined in [RFC5246] Section 7.4.1.2.

server_random  The random value associated to the TLS Server as defined in [RFC5246] Section 7.4.1.2.

tls_version  The TLS version set by the TLS Client as defined in [RFC5246] Section 6.2.1.

EncryptedPreMasterSecret  The encrypted master secret as defined in [RFC5246] Section 7.4.7.1.

5.1.1.  PFS-PRFA

This document defines a mechanism based on a one way functions to prevent an attacker to send a request to the LURK Server to replay an observed TLS handshake. More explicitly, this document assumes that the ServerHello.random of the TLS exchange is derived from the server_random of the LURK exchange as defined below:

### with PFS-PRFA != "null"

gmt_unix_time = server_random[0..3];
ServerHello.random = PFS-PRFA( server_random + "tls12 pfs" );
ServerHello.random[0..3] = gmt_unix_time;

### with PFS-PRFA == "null"

ServerHello.random = server_random;

The server_random MUST follow the structure of [RFC5246] section 7.4.1.2 which carry the gmt_unix_time in the first four bytes.

The operation MUST be performed by the LURK Server as well as the TLS Server, upon receiving the master secret or the signature of the ecdhe_params from the LURK Client. When PFS-PRFA is different from "null", then the server_random cannot be derived from the observed ServerHello.random, which prevents the handshake to be replayed.

Importantly a LURK Server implementing PFS MUST NOT enable the "null" PFS-PRSA.

5.2.  Response Payload

The "rsa_master" response payload contains the master secret and has the following structure:

struct {
  opaque master[0..47];
} TLS12RSAMasterResponsePayload;
5.3. LURK Client Behavior

A rsa_master request is sent so the LURK Server can derive the master secret used by the TLS session. Upon receipt of the master_secret the Edge Server can generate the session keys and finish the TLS key exchange protocol.

5.4. LURK Server Behavior

Upon receipt of a rsa_master request, the LURK Server proceeds according to the following steps:

(1) Checking TLS12Base:

(a) The LURK Server checks the RSA key pair is available (key_id). If the format of the key pair identifier is not understood, an "invalid_key_id_type" error is returned. If the designated key pair is not available an "invalid_key_id" error is returned.

(b) The LURK Server checks the length of the encrypted premaster secret and so the length of the payload. In case of length mismatch and "invalid_payload_format" error is returned. Note that the length can be derived from the public key, so the error only reveals public information.

(c) If the TLS version is not supported or is different from TLS 1.1 or TLS 1.2 an "invalid_tls_version" error is returned.

(d) The LURK Server MAY check the format of the random, and more specifically checks the gmt_unix_time associated to the random is acceptable. Otherwise it SHOULD return an "invalid_tls_random" error. The value of the time window is implementation dependent and SHOULD be a parameters and SHOULD be configurable.

(2) The LURK Server checks the prf. If it does not support the PRF, an "invalid_prf" error is returned.

(3) The LURK Server computes the necessary ServerHello.random from the server_random when applicable as described in Section 5.1.1.

(4) The LURK Server decrypts the encrypted premaster secret as described in [RFC5246] section 7.4.7.1. When a PKCS1.5 format error is detected, or a mismatch between the TLS versions provided as input and the one indicated in the encrypted
premaster secret, the Key Server returns a randomly generated master secret.

(5) If the pre master is appropriately decrypted, then the master secret is computed as described in [RFC5246] section 8.1 using the MS-PRFA, the client_random, and the server_random provided by the LURK Client.

(6) The LURK Server returns a master secret in a TLS12RSAMasterResponsePayload.

(7) Error are expected to provide the LURK Client an indication of the cause that resulted in the error. When an error occurs the LURK Server MAY ignore the request, or provide more generic error codes such as "undefined_error" or "invalid_format".

6. rsa_extended_master

A exchange of type "rsa_extended_master" enables the LURK Client to delegate the RSA Key Exchange and authentication. The LURK Server returns the extended master secret as defined in [RFC7627].

6.1. Request Payload

The "rsa_extended_master" payload request has the following structure:

```c
struct{
    KeyPairID key_id
    ProtocolVersion tls_version   // see RFC5246 section 6.2.1
    PRFAlgorithm prf              // see RFC5246 section 6.1
    EncryptedPreMasterSecret  pre_master
    // see RFC5246 section 7.4.7.1
    opaque session_hash<2...2^16-2>
}TLS12ExtendedMasterRSARequestPayload;
```

key_id, tls_version, prf, pre_master are defined in Section 5.1.

session_hash The hash of the TLS handshake session as described in [RFC7627] section 3.

Note that as the server_random is not provided by the LURK Client, PFS mechanisms does not apply here, and only the PFS-PRFA "null" can be used.
6.2. Response Payload

rsa_extended_master response payload has a similar structure as the
rsa_master response payload Section 5.2.

6.3. LURK Client Behavior

A rsa_extended_master request is sent so the LURK Server can derive
the master secret used by the TLS session. Upon receipt of the
master_secret the Edge Server can generate the session keys and
finish the TLS key exchange protocol.

6.4. LURK Server Behavior

The LURK Server proceeds as described in Section 5.4 except that the
generation of the extended master is processed as described in
[RFC7627].

7. ecdhe

A exchange of type "ecdhe" enables the LURK Client to delegate the
ECDHE_RSA [RFC5246] or the ECDHE_ECDSA [RFC4492] authentication.

7.1. Request Payload

The "ecdhe" request payload has the following structure:

enum { null(0), sha256_128(1), sha256_256(2),
(255) }POOPRF

struct {
    POOPRF poo_prf;
    select( poo_prf) {
        case ( "null" ):  
        case ( "sha256_128" or "sha256_256" ): 
            ECPoint rG;  //RFC4492 section 5.4
            ECPoint tG;
        }
    } TLS12POOParams;

struct {
    TLS12Base base;
    ServerECDHParams ecdhe_params;  // RFC4492 section 5.4
    POOParams poo_params;
} TLS12ECDHERequestPayload;

base is defined in Section 5.1.
ecdhe_params contains the ECDHE public key - also noted bG - and associated domain parameters as defined in [RFC4492] section 5.4. In the notation bG for the ECDHE public, b is a random secret generated by the LURK Client and G the base point of the curve. The base is explicitly specified in the ECP Barrett, 2018 structure for curves of type "explicit_prime" or "explicit_char2". For curves of type "named_curves" G is part of the definition of the curve.

poo_params defines the necessary parameters to provide a proof of ownership of the ECDHE private key. This option is intended to prevent the LURK Server to sign bytes that do not correspond to a ECDHE public key.

poo_prf pseudo random function used to generate the necessary randoms to proof ownership of the private key. This document defines sha256_128 and sha256_256 which apply the sha256 hash function and respectively return the 128 or 256 first bits of the resulting hash.

rG, tG are necessary points to generate the proof of ownership. r is a random number chosen by the LURK Client. G is the the base point of the curve. t = cb + r, with c a number that is not under the control of the LURK Client generated as the output of poo_prf and b the random secret of the private key.

The proof of ownership consists in the LURK Client proving the knowledge of the private random b, while not disclosing b. c MUST NOT be under the control of the LURK Client. To achieve that goal, c is generated as described below:

c = poo_prf ( base + ecdhe_params + "tls12 poo")

The LURK Client computes t = cb + r and sends rG, tG, in poo_params and bG in the ecdhe_params

The LURK Server computes c(bG) + rG and compares the output with tG. The equality proves the ownership of b by the LURK Client.

Note r and c may be treated as "very short-term secrets" but MUST remain non-predictable. It is RECOMMENDED to use a length equivalent to the expected level of security, that is 128 bit length (resp. 256 bit length) for a 128 (resp 256) bit security level. Given b, we RECOMMEND r and c to be at least half the size of b.
7.2. Response Payload

The "ecdhe" response payload has the following structure:

```c
struct {
    Signature signed_params; // RFC4492 section 5.4
} TLS12ECDHEResponsePayload;
```

signed_params signature applied to the hash of the ecdhe_params as well as client_random and server_random as described in [RFC4492] section 5.4.

7.3. LURK Client Behavior

The LURK Client builds the base as described in Section 5.1. The LURK Client computes c, t, rG and tG as described in Section 7.1.

Upon receiving the response payload, the LURK Client MAY check the signature. If the signature does not match an error SHOULD be reported.

7.4. LURK Server Behavior

Upon receiving an ecdhe request, the LURK Server proceeds as follows:

1. The LURK Server performs some format check of the ecdhe_params before signing them. If the ecdhe_params does not follow the expected structure. With the notations from [RFC4492], if curve_type (explicit_prime, explicit_char2 or named_curve) is not supported, the LURK Server SHOULD be able to respond with an "invalid_ec_type" error. With curve of type "explicit_char2", if the basis (ec_basis_triomial or ec_basis_pentanomial) is not supported, the LURK Server SHOULD be able to respond with an "invalid_ec_basis_type" error. If the curve or namedcurve is not supported the LURK Server SHOULD be able to respond with an "invalid_ec_curve" error. For curves of type explicit_prime or explicit_char2 when the expression of the point is not supported, the LURK Server SHOULD be able to respond with an "invalid_ec_point_format" error.

2. The LURK Server processes the poo_params. If the poo_prf is not supported, the LURK Extension returns a "invalid_poo_prf" status. If poo_prf is supported and different from "null", the LURK Server proceeds to the proof of ownership as described in Section 7.1. If the proof is not properly verified, the LURK Extension returns a "invalid_poo" status.
The LURK Server processes the base structure as described in Section 5.4.

The LURK Server generates the siged_params.

Error are expected to provide the LURK Client an indication of the cause that resulted in the error. When an error occurs the LURK Server MAY ignore the request, or provide more generic error codes such as "undefined_error" or "invalid_format".

8. capabilities

A exchange of type "capabilities" enables the LURK Client to be informed of the supported operations performed by the LURK Server. The supported parameters are provided on a per type basis.

8.1. Request Payload

A LURK "capabilities" request has no payload.

8.2. Response Payload

The "capabilities" response payload lists for each supported type, the supported certificates, the supported signatures and hash associated. The "capabilities" payload has the following structure:
struct{
    KeyPairIdType key_id_type_list<0..255>;
    ProtocolVersion tls_version_list<0..255>;
    PRFAlgorithm prf_list<0..255> // [RFC-TBD] rsa_master section
    CertificateEntry certificate_list <0..2^26-2>;
    // draft-ietf-tls-tls13 section 4.4.2
}TLS12BaseCapabilities

struct{
    TLS12BaseCapabilities base_capabilities;
    SignatureAndHashAlgorithm algo_list<2..2^16-2>;
    // RFC5246 section 7.4.1.4.1
    POOPRF poo_prf_list <0..255>;
    NamedCurveList supported_curve_list
    //draft-ietf-tls-rfc4492 section 5.1.1
    ECPointFormatList ec_format_list
    //draft-ietf-tls-rfc4492 section 5.1.2
}TLS12ECDHECapabilitites

struct {
    TLS12Type type_list<0..255>;
    TLS12BaseCapabilities rsa_capabilities;
    TLS12BaseCapabilities extended_rsa_capabilities;
    TLS12ECDHECapabilitites ecdhe_capabilities;
    opaque state<32>;
}TLS12CapabilitiesResponsePayload;

key_id_type_list  the supported key_id_type.

tls_version_list  the supported tls_versions. The format is defined in [RFC5246] section 6.2.1.

prf_list designates the list of prf ( see Section 5.1). As prf indicates whether PFS is supported and that PFS is globally supported for a given type only if all prf support PFS, the list enables the LURK Client to check PFS is appropriately implemented for a given type.

certificate_list designates the certificates associated to message type. The format is defined in [I-D.ietf-tls-tls13] in section 4.4.2. This format enables the use of X509 as well as Raw Public key, while the Certificate structure defined in [RFC5246] section 7.4.2 do not.
algo_list designates supported signature algorithms as well as PRF used for the different operations. The format is defined in [RFC5246] section 7.4.1.4.1.

poo_prf_list the supported message type poo_prf (see Section 7.1. to be used with the proof of ownership.

type_list the supported message type of the LURK extension.

state characterizes the configuration associated to ‘tls12’ on the LURK Server.

8.3. LURK Client Behavior

The LURK Client performs a capability request in order to determine the possible operations.

The LURK Client is expected to keep the state value to be able to detect a change in the LURK Server configuration when an error occurs.

8.4. LURK Server Behavior

Upon receiving a capabilities request, the LURK Extension MUST return the capabilities payload associated to a "success" status to the LURK Server. These information are then forwarded by the LURK Server to the LURK Client.

9. ping

A exchange of type "ping" enables the LURK Client to check the reachability in a context of the defined LURK Extension.

9.1. Request Payload

A "ping" request has no payload.

9.2. Response Payload

A "ping" response has no payload.

9.3. LURK Client Behavior

The LURK Client sends a "ping" request to test the reachability of the LURK Server. The reachability is performed for the tls12 LURK Extension.
9.4. LURK Server Behavior

Upon receiving a ping request, the LURK Extension MUST return the ping response associated with a "success" status to the LURK Server. These information are then forwarded by the LURK Server to the LURK Client.

10. Security Considerations

The security considerations defined in [I-D.mglt-lurk-lurk] applies to the LURK Extension "tls12" defined in this document.

The LURK Extension "tls12" is expected to have response smaller that the request or at least not significantly larger, which makes "tls12" relatively robust to amplification attacks.

The LURK Extension "tls12" performs cryptographic operations that may lead to resource exhaustion, as such it is RECOMMENDED the LURK Server authenticates the LURK Clients. In addition, this prevents when UDP is used the LURK Server to be used as a reflector.

The rsa_master and ecdhe exchange provides a mechanism to provide Perfect Forward Secrecy. It is RECOMMENDED to activate such mechanism and disable any PFRAlgorithm suites that contains a PFS-PFR set to null. Enabling a single PFS-PRF MAY compromise the PFS property.

The rsa_extended_master does not provide PFS mechanism as a digest of the TLS Key Exchange is provided which prevent the LURK Server to perform an agreed operation on the HelloServer.random.

rsa_master and ecdhe request provides client_random. In order to limit the replay of a random value it RECOMMENDED the Server control the time at which the random has been generated. The time control should also consider desynchronization between the TLS Client and the LURK Server.

10.1. RSA

The rsa_master and rsa_extended_master exchange carry the master secret. These pieces of information enable a passive attacker to hijack the session and as such it is RECOMMENDED the LURK Server and the LURK Client use an encrypted channel when the authentication RSA is used.

The rsa_master and rsa_extended_master returns the master_secret instead of the premaster. The additional hashing operation necessary to generate the master secret is expected to improve the protection
of the RSA private key against cryptographic analysis based on the observation of a set of clear text and corresponding encrypted text.

Note that having the Key Server generating the server random on its own without any input from the Edge Server would also achieve the PFS in a similar way as standard TLS does. The pfs options achieves similar properties while enabling a LURK Client to replay a request.

The standard TLS1.2 is robust against Bleichenbacher attack as it provides no means to detect if the error comes from a TLS version mismatch or from the premaster format. This properties remain with LURK, and so LURK does not present vulnerabilities toward Bleichenbacher attack, and cannot be used as a decryption oracle.

10.2. ECDHE

A passive attacker observing the ecdhe exchange may collect a sufficient amount of clear text and corresponding signature to perform a cryptographic analysis or to reuse the signature for other purposes. As a result, it is RECOMMENDED to encrypt the ecdhe exchange between the LURK Client and the LURK Server. Note that this vulnerability is present in TLS 1.2 as a TLS Client can accumulate these data as well. The difference with LURK is by listening the LURK Server, the accumulation is achieved for all TLS Clients.

As previously mentioned, the LURK Server may be used as signing oracle for the specific string:

SHA(ClientHello.random + ServerHello.random + ServerKeyExchange.params);

More specifically, the ECDHE_RSA and ECDHE_DSA mechanisms does not associate the signature to a TLS1.2 context. As a result, the signature could be re-used in another context.

The attack may operate by collecting a large collection of clear text and their corresponding signature. When the attacker want to provide a signature, it checks in its database, a match occurs between the two contents to be signed. The probability of a collision increases with number of available hashes. The attack is related the pre-image and collision resistance properties of the hash function.

The attacker may also given a clear text to be signed, generate a collision such that a collision occurs which provides is related to the second pre-image and collision resistance property of the hash function.

The surface of attack is limited by:
a. limiting the possibility of aggregating a collection of clear
text and their corresponding signatures. This could be achieved
by using multiple LURK Clients using an encrypted channel between
the LURK Client and the LURK Server.

b. increasing the checks and ensure that signature is performed in a
tLS 1.2 context. For that purpose it is RECOMMENDED the LURK
Server checks the consistency of its input parameters. This
includes the proof of ownership as well as the format of the
randoms and ecdhe_params for example.

c. limiting the usage of a Cryptographic material to a single usage,
in our case serving TLS 1.2.

11. IANA Considerations

The requested information is defined in [I-D.mglt-lurk-lurk].
LURK Extension Designation: tls12
LURK Extension Reference: [RFD-TBD]
LURK Extension Description: RSA, ECDHE_RSA and ECDHE_ECDSA for (D)TLS 1.1 and (D)TLS 1.2.

LURK tls12 Extension Status

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 1</td>
<td>Reserved</td>
<td>[RFC-TBD-LURK]</td>
</tr>
<tr>
<td>2</td>
<td>undefined_error</td>
<td>[RFC-TBD]</td>
</tr>
<tr>
<td>3</td>
<td>invalid_payload_format</td>
<td>[RFC-TBD]</td>
</tr>
<tr>
<td>4</td>
<td>invalid_key_id_type</td>
<td>[RFC-TBD]</td>
</tr>
<tr>
<td>5</td>
<td>invalid_key_id</td>
<td>[RFC-TBD]</td>
</tr>
<tr>
<td>6</td>
<td>invalid_tls_version</td>
<td>[RFC-TBD]</td>
</tr>
<tr>
<td>7</td>
<td>invalid_tls_random</td>
<td>[RFC-TBD]</td>
</tr>
<tr>
<td>8</td>
<td>invalid_prf</td>
<td>[RFC-TBD]</td>
</tr>
<tr>
<td>9</td>
<td>invalid_encrypted_premaster</td>
<td>[RFC-TBD]</td>
</tr>
<tr>
<td>10</td>
<td>invalid_ec_type</td>
<td>[RFC-TBD]</td>
</tr>
<tr>
<td>11</td>
<td>invalid_ec_basis_type</td>
<td>[RFC-TBD]</td>
</tr>
<tr>
<td>12</td>
<td>invalid_ec_curve</td>
<td>[RFC-TBD]</td>
</tr>
<tr>
<td>13</td>
<td>invalid_ec_point_format</td>
<td>[RFC-TBD]</td>
</tr>
<tr>
<td>14</td>
<td>invalid_poo_prf</td>
<td>[RFC-TBD]</td>
</tr>
<tr>
<td>15</td>
<td>invalid_poo</td>
<td>[RFC-TBD]</td>
</tr>
<tr>
<td>16 - 255</td>
<td>UNASSIGNED</td>
<td>[RFC-TBD]</td>
</tr>
</tbody>
</table>

LURK tls12 Extension Type

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>capabilities</td>
<td>[RFC-TBD]</td>
</tr>
<tr>
<td>1</td>
<td>ping</td>
<td>[RFC-TBD]</td>
</tr>
<tr>
<td>2</td>
<td>rsa_master</td>
<td>[RFC-TBD]</td>
</tr>
<tr>
<td>3</td>
<td>rsa_extended_master</td>
<td>[RFC-TBD]</td>
</tr>
<tr>
<td>4</td>
<td>ecdhe</td>
<td>[RFC-TBD]</td>
</tr>
<tr>
<td>16 - 255</td>
<td>UNASSIGNED</td>
<td>[RFC-TBD]</td>
</tr>
</tbody>
</table>

12. Acknowledgments

We would like to thank for their very useful feed backs: Yaron Sheffer, Yoav Nir, Stephen Farrell, Eric Burger, Thomas Fossati, Eric Rescorla Mat Naslung, Rich Salz. Many ideas in this document are from [I-D.erb-lurk-rsalg].

We would also like to thank those that have supported LURK or raised interesting discussions. This includes among others Robert Skog, Hans Spaak, Salvatore Loreto, John Mattsson, Alexei Tumarkin, Yaron
13. References

13.1. Normative References


13.2. Informative References


Appendix A. Example of LURK Exchanges
A.1. LURK/TLS RSA Master Secret

TLS Client          Edge Server         Key Server

ClientHello
    server_version
    client_random
    cipher_suite
        TLS_RSA_*, ...
-------->

ServerHello
    tls_version
    server_random
    Cipher_suite=TLS_RSA

Certificate
    RSA Public Key

ServerHelloDone
<---------

ClientKeyExchange
    EncryptedPremasterSecret
[ChangeCipherSpec]

Finished
-------->

TLS12 Request Header
TLS12MasterRSARequestPayload
    key_id
    client_random
    edge_server_random
    tls_version
    master_prf
    EncryptedPremasterSecret
-------->

1. Computing Master Secret
master_secret = master_prf(
    pre_master_secret + \\n    "master_secret" +\\
    client_random +\\
    edge_server_random)[0..47];

TLS12 Response Header
TLS12MasterResponsePayload
    master
<--------
A.2. LURK/TLS RSA Extended Master Secret
TLS Client          Edge Server         Key Server
ClientHello
    tls_version
cipher_suite
        TLS_RSA_*, ...
    Extension 0x0017
-------->
    ServerHello
        edge_server_version
cipher_suite=TLS_RSA
    Extension 0x0017
Certificate
    RSA Public Key
ServerHelloDone
<--------
ClientKeyExchange
    EncryptedPremasterSecret
[ChangeCipherSpec]
Finished
-------->
    TLS12 Request Header
    TLS12ExtendedMasterRSAInputPayload
        key_id
tls_version
master_prf
session_hash
    EncryptedPreMasterSecret
-------->
1. Computing Master Secret
    master_secret = master_prf(
        pre_master_secret +
        "extended master secret" +
        session_hash)[0..47]
    TLS12 Response Header
    TLS12MasterPayload
    master
<--------
[ChangeCipherSpec]
    Finished
<-------- Application Data      <------>
Application Data
A.3. LURK/TLS ECDHE Signature

TLS Client          Edge Server         Key Server

ClientHello
  tls_version
  client_random
  cipher_suite
    TLS_ECDHE_ECDSA_*, TLS_ECDHE_RSA_*, ...
  Extension Supported EC, Supported Point Format

-------->

TLS12 Request Header
  TLS12ECDHEInputPayload
    key_id
      client_random
      edge_server_random
    tls_version
    ecdhe_params

-------->

1. Generating the signature
signatures = ECDSA(client_random +
                edge_server_random + ecdhe_params)

TLS12 Response Header
  TLS12DigitallySignedPayloads
    signature

<--------

ServerHello
  edge_server_version
  edge_server_random
  Cipher_suite=TLS_ECDHE_ECDSA
  Extension Supported EC,
  Supported Point Format

Certificate
  ECDSA Public Key
ServerKeyExchange
  ecdhe_params
  signature
ServerHelloDone

<--------

ClientKeyExchange
  [ChangeCipherSpec]
Finished

-------->
[ChangeCipherSpec]
Finished
<--------
Application Data  <-------->  Application Data

Author’s Address

Daniel Migault (editor)
Ericsson
8400 boulevard Decarie
Montreal, QC  H4P 2N2
Canada

Phone: +1 514-452-2160
Email: daniel.migault@ericsson.com