Lurk - BoF

Daniel Migault

Toc

- TLS Authentication with LURK
  - Case of RSA
  - Case of ECDHE_ECDSA
- LURK Protocol
  - LURK Cryptographic Exchange
  - LURK Implementation
- LURK drafts

TLS Authentication with LURK

```
+-------------+         +------------------------------------------+
|TLS Client   |         | TLS Server                               |
+-------------+         +------------------------------------------+
|Web Browser  |         | Edge Server               Key Server     |
|             |         |                           (Private Key)  |
+-------------+         +------------------------------------------+

ClientHello -------->
    Cipher_suite

                  <------- ServerHello
                Cipher_suite (Selected)

                         <============= LURK ==============>

--- [Remaining TLS Handshake] ---

Cipher_suite examples:
    - TLS_RSA_WITH_AES_256_CBC_SHA256
    - TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
    - TLS_DH_RSA_WITH_AES_256_CBC_SHA256
    - TLS_DH_anon_WITH_AES_256_CBC_SHA25
    - TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
```

TLS Authentication with LURK

- TLS authentication methods:
  - RSA
  - DH_DSS, DH_RSA, ECDH_ECDSA, ECDH_RSA
  - DH_anon, ECDH_anon
  - DHE_DSS, DHE_RSA, ECDHE_RSA, ECDHE_ECDSA
  - PSK, DHE_PSK, RSA_PSK

RSA Model

```
 Web Browser            Edge Server                Key Server
                                                 (RSA private Key)
        <------------   ServerHello
                        ServerCertificate
                        (RSA Public Key)

 ClientKeyExchange  -------->
    EncryptedPremasterSecret

                        EncryptedPremasterSecret ------------>
                        None  OR PRF      OR PRF          OR session_hash
                              | client.R    |  handshake    |
                              | server.R    |
                                        (pMS = decrypt(EncryptedPremasterSecret))
                                        (MS = PRF(pMS, + “MS“+ client.R + server.R))

                                        (EMS = PRF(pMS, + “EMS” + session_hash))

                              <------------  pMS / MS / EMS

[(Compute MS/EMS)]        [(Compute MS/EMS)]
(Derive Session Keys)     (Derive Session Keys)
```

Discussion: Limiting Outputs to (E)MS

- Returning a pre Master Secret:
  - Pro: Universal interface as MS/EMS are derived from the premaster
  - Con: PMS makes the Key Server an open oracle

- Returning a Master Secret / Ext MS:
  - Pro: needs some context of a TLS connection establishment
  - Pro: Keep the premaster secret from the Edge Server
    - Would reduce vulnerability in case the premaster is re-used.
  - Pro: Protects the Key Server from chosen cipher text attack, and leakage of RSA private key
  - Con: requires different implementation for the MS/EMS

- Any opinion on limiting outputs to MS and EMS ?

Discussion: Limiting EMS Input to sH

- ~~Output is premaster:~~
  - ~~Encrypted Premaster Master~~
- Output is Master Secret:
  - Encrypted Premaster Master
  - PRF
  - {Server/Client}Hello.random
- Output is Extended Master Secret
  - Encrypted Premaster Master
  - PRF
  - Session_hash

  OR

  - ~~Encrypted Premaster Master~~
  - ~~PRF~~
  - ~~Handshake_message~~
  - ~~Hash~~

- Which inputs for the EMS:
  - Session_hash
    - Pro: limited overhead
    - Con: provides little TLS context (opaque value)
  - handshake_message
    - Pro: provides a better context
    - Con: significant network overhead
    - Con: significantly complex parameters checks at the Key Server

- Any opinion on only considering the session_hash ?

ECDHE_ECDSA

```
Web Browser            Edge Server               Key Server
                                                (ECDSA private Key)

ClientHello     ------------>
Supported Elliptic Curves Extension
Supported Point Formats Extension
                              ServerECDHParams  OR hash -------->
                              client.R           |
                              server.R           |
                              H                  |
                                                   H(client.R + server.R + ECDHParams)
                                               <--------- Signature

                 <----------- ServerHello
                                  Supported Point Formats Extension
                              Certificate
                              ServerKeyExchange
                                  ECParameters (ECDH Public key)
                                  Signature (ECDSA)


 ClientKeyExchange --------->
 ClientECDiffieHellmanPublic (ECDH Public key)
 (Compute ECDH)                (Compute ECDH)
 (Compute EMS/ MS)             (Compute MS / EMS)
```

Discussion: Input Parameters

- Input can be:
  - Hash function
  - ServerHello.random
  - ClientHello.random
  - ServerKeyExchange.params

  OR
  - Resulting hash
- Any opinion on providing all parameters?

Discussion: Output Signature Format

- TLS1.3 proposes DigitallySigned structure

```
struct {
    SignatureScheme algorithm;
    opaque signature<0..2^16-1>;
} DigitallySigned;
```

- TLS1.3 propose to replace RSASSA-PKCS1-v1_5 by RSA-PSS
- Any opinion on adopting TLS1.3 structures and format for LURK?

LURK Protocol

- LURK Implementation design:
  - Minimal MTI set of authentication methods (ECDHE)
  - Extensible for any other authentication method

- Notes:
  - None of the authentication method is incompatible with LURK
  - Minimal MTI should consider:
    - Non deprecated authentication methods
    - Deployed
    - We can still update later and deprecate non used authentication methods

LURK Protocol

- Type of Exchange:
  - – Cryptographic Oriented Exchanges
  - – Additional Exchange
    - Edge Server Listing Capabilities (supported extensions)
    - Keep Alive
    - Extra Information provided to the Edge Server
      - – Checking the public key hosted on the Edge Server
    - …

LURK Cryptographic Exchange

- LURK Client Input parameters:
  - Static parameters
    - LURK version
    - TLSVersion
    - ObjectRequest:
      - ~~premaster~~ / master / signature
    - AuthenticationMethod
      - rsa, ~~dh_dss, dh_rsa, dh_dss, dh_rsa, ecdh_rsa~~, ~~dh_anon~~, ~~ecdh_anon, dhe_dss~~, dhe_rsa, ecdhe_ecdsa, ecdhe_rsa, ~~psk, dhe_psk, rsa_psk~~
    - MasterMethod
      - ms, emas_from_session_hash, ~~ems_from_handshake~~
    - SignatureMethod
      - rsa, ~~dss~~, ecdsa
  - PRF
  - TLS Handshake parameters
    - Client/ServerHello.random
    - Session_hash, ~~data_hash~~
    - ~~Handshake_message~~
    - ~~PSK_id~~
  - Cryptographic parameters
    - RSAEncryptedPreMaster
    - DHECDHEEdgeServerDHEECDHEParams
    - TLSClientDHECDHPublicKey
    - Should we indicate the private key concerned ?

LURK Cryptographic Exchange

- LURK Client Output parameters:
  - ~~PreMaster~~ :

```
opaque random[46]
```

  - Master/EMaster:

```
opaque random[48]
```

  - Signature

```
struct {
     SignatureScheme algorithm;
     opaque signature<0..2^16-1>;
} DigitallySigned;
```

  - Error

LURK Cryptographic Exchange

- Type of Error:
  - Incoherence between Input parameters:
    - Ex: authentication method = rsa and object request = signature
    - ~~TLSVersion and RSAEncryptedPreMaster~~
  - Unsupported Input parameters
    - Ex: authentication method = psk
  - Operations Errors:
    - Should not be provided (decryption)
    - It should be logged
    - DOTS should generate an alarm (outside LURK)

LURK Implementation

- A single implementation:
  - Current structures are really TLS oriented
  - HTTPS/JSON ?
  - HTTPS/CBOR ?
  - New packet format ?

LURK drafts

- draft-mglt-lurk-tls-use-cases
  - describes the use case
- draft-mglt-lurk-tls-requirements
  - ALL authentication method
  - Describe Split Authentication
  - Provide Security Requirements for Split Authentication
  - Provides a Security Analysis of Split Authentication
- draft-cairns-tls-session-key-interface
  - Describes the architecture
  - Implementation for RSA/(EC)DHE* based on JSON
- draft-mglt-lurk-tls-abstract-api
  - Describes interactions between the Edge Server / Key Server
  - Does not describe the payload format… yet!

Thank you!