A Formal TLS Handshake Model in LNT

Josip Bozic, Franz Wotawa
Graz University of Technology
Institute of Software Technology
8010 Graz, Austria
{jbozic,wotawa}@ist.tugraz.at

Lina Marsso, Radu Mateescu
Univ. Grenoble Alpes, Inria,
CNRS, Grenoble INP, LIG
38000 Grenoble, France
{lina.marsso,radu.mateescu}@inria.fr

MARS/VPT, April 20^{th}, 2018

Outline

1. Introduction

2. Formal model

3. Validation

4. Conclusion

Introduction

- Security services in e-government, online banking, online shops, social media, …
- New vulnerabilities are detected on a regular basis.
- Many faults have their roots in the software development cycle or intrinsic leaks in the system specification.
- Testing of network services represents one of the biggest challenges in cyber security.
- Conformance testing checks whether a system behaves according to its specification.
- Formal specification of a system behavior.

Contributions

- Formalization of the Handshake protocol of the Transport Layer Security (TLS) in the LNT language.
- Conformance testing of TLS implementations.
- Connection to framework for automated testing of TLS implementations [1].

Transport Layer Security (TLS)

- Security/cryptographic protocols assure reliable and secure communication between peers.
- Predecessor of TLS: the Secure Sockets Layer (SSL).
- Currently used version: TLS 1.2 [3];
  Working draft: TLS 1.3.
- Reasons for vulnerability: Complexity of the protocol and its high number of interactions.

Known TLS Vulnerabilities

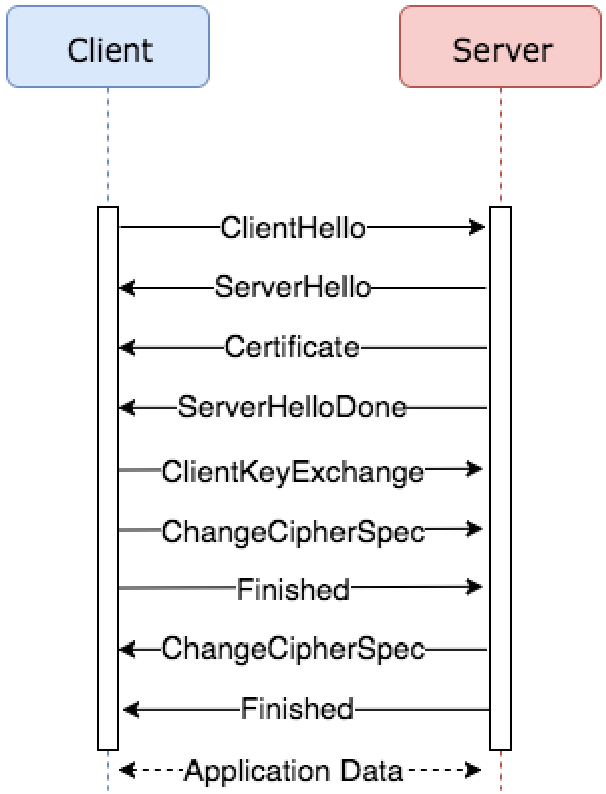
- BEAST (CVE-2011-3389)
- CRIME (CVE-2012-4929)
- BREACH (CVE-2013-3587)
- Heartbleed (CVE-2014-0160)
- POODLE (CVE-2014-3566)
- DROWN (CVE-2016-0800):
  33% of all HTTPS sites were affected [4].

Vulnerabilities of implementations (not the protocol).

TLS Handshake Protocol

- One of the most complex and vulnerable parts of TLS.
- Consists of TLS messages.
- Every of these messages encompasses a specific set of parameters and values.

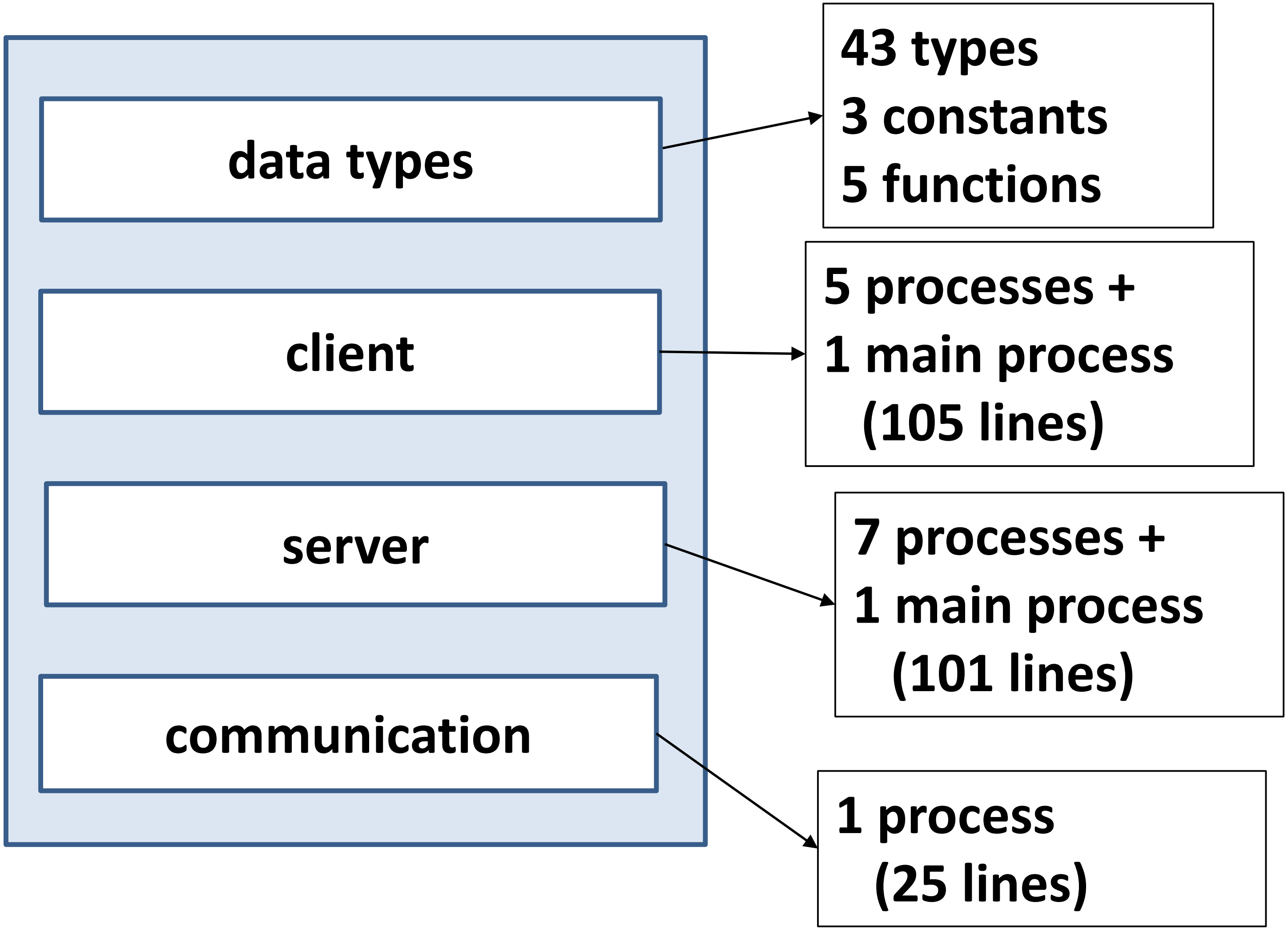
➢ Our task: Implement the interaction and execute it for testing purposes.



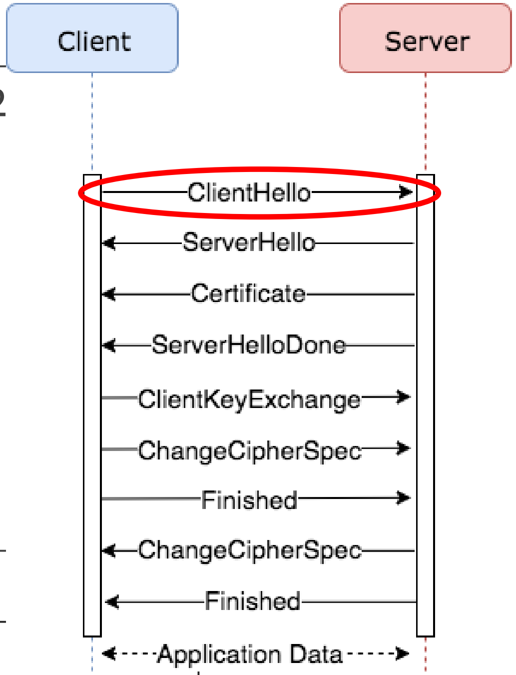
Formal Model of TLS 1.3 Handshake

- LNT
  - Formal specification language for concurrent systems.
  - Process calculus with imperative syntax.
  - Imperative language.
- Starting point
  - Description of state machines [draft-tls-1.3].
  - TLS 1.3 handshake informal requirements
    (not self-contained: refers to further documents).

Model Overview



Data Type Example: ClientHello (1/2)




```
Protocol Version : TLS10, TLS11, TLS12, DTLS10, DTLS12
Client Random : 28-byteRand
Session ID : NULL, 32-byteID
Supported Cipher Suites : TLS_FALLBACK_SCSV,
TLS_NULL_WITH_NULL_NULL, TLS_RSA_WITH_NULL_SHA256,
TLS_RSA_WITH_AES_128_CBC_SHA256,
TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA
Supported Compression Methods : NULL, DEFLATE, LZS
Extensions : extension_type, extension_data
```

```
type ClientHello is
   ClientHello (legacy_version: ProtocolVersion,
                random: Random32, legacy_session_id: SessionId,
                cipher_suite: Ciphers,
                legacy_compression_methods: CompressionMethods,
                extensions: Extensions)
end type
```

Data Type Example: ClientHello (2/2)

```
var e: Extensions in
   e := {Extension (supported_version, SupportedVersion ({TLS12})) }
end var
```

```
type Extensions is
   list of Extension
   with "member", "remove"
end type
```

```
type ExtensionType is
   signature_algorithms,
   supported_versions,
   ...
end type
```

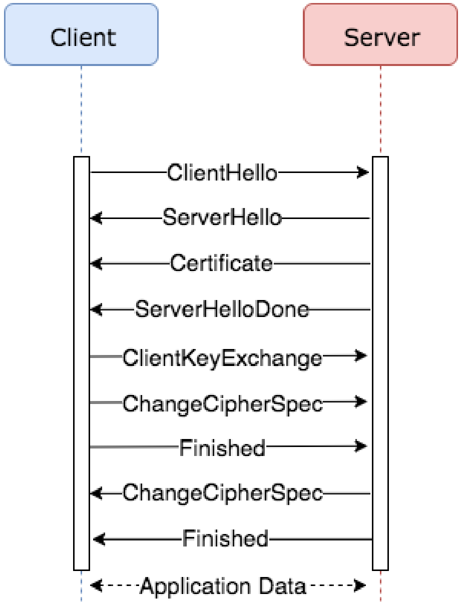
```
type Extension is
   Extension (
      type: ExtensionType,
      data: ExtensionData)
end type
```

```
type ExtensionData is
   Cookie (c: Cookie),
   SupportedVersions (
    sv: supportedVersion),
   ...
end type
```

```
type SupportedVersions is
   list of  ProtocolVersion
end type
```

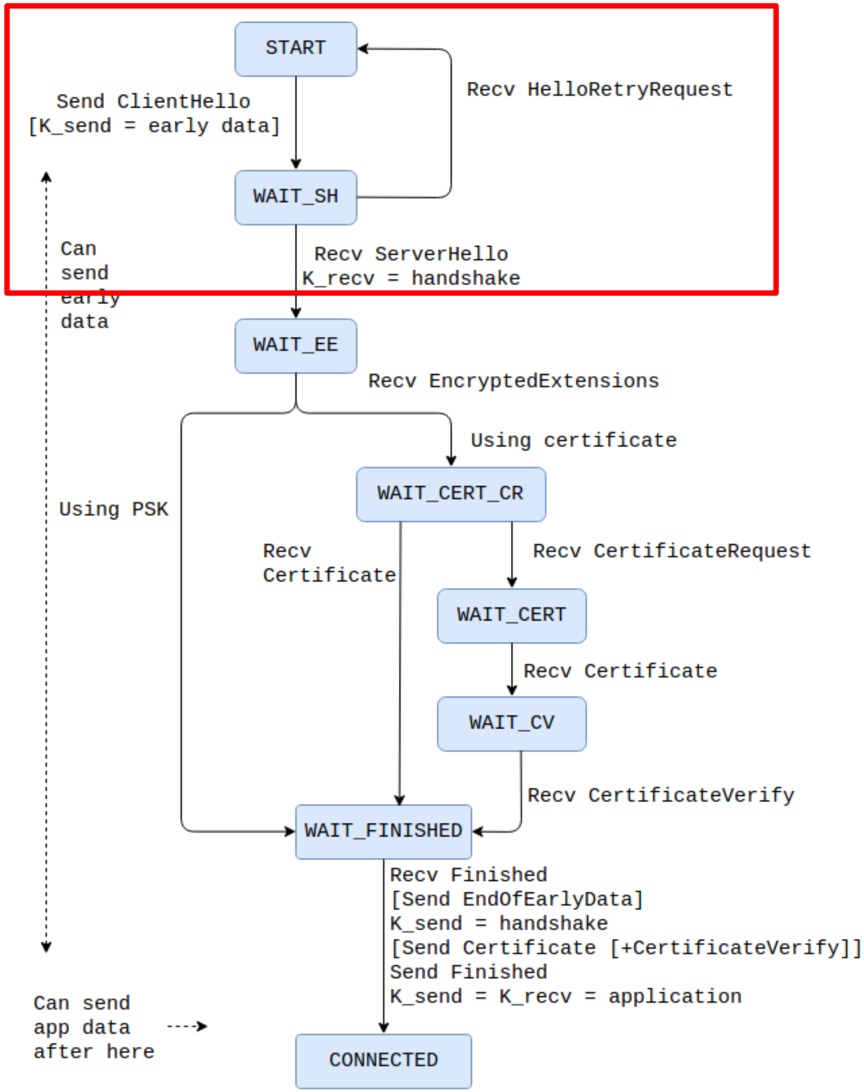
Client, Server and their Interactions

- Interactions described by sequence diagrams.
- Incomplete state machines for client and server
  - Human readable.
  - Compact.



\+ Added management of Alerts
  - Handling handshake errors.
  - Requirements not respected.

(incomplete) Client-side State Machine




```
-- Start
loop L in
  -- client key exchange [K_send = early data]
  ClientHello [clientHello_c] (is_helloRequest, !?CH_p, HRR_P,
?alert);
  if alert != undefined then
    -- abort the handshake with an alert
    alert_c (alert)
  else  -- WAIT_ServerHello
    select
      helloRetryRequest_c (?HRR_P);
      is_helloRequest := true
    [] serverHello_c (?any ServerHello);
      break L
    [] -- protocol messages sent in the wrong order
     select
       encryptedExtensions_c (?any EncryptedExtensions)
      [] certificateRequest_c (?any CertificateRequest)
      …
     end select;
     alert := unexpected_message;
     -- abort the handshake with an "unexpected_message" alert
     alert_c (alert)
    end select
  end if
end loop;
```

TLS Interruptions

Informal requirements

- "The TLS 1.3 handshake refuses renegotiation without a hello retry request message."

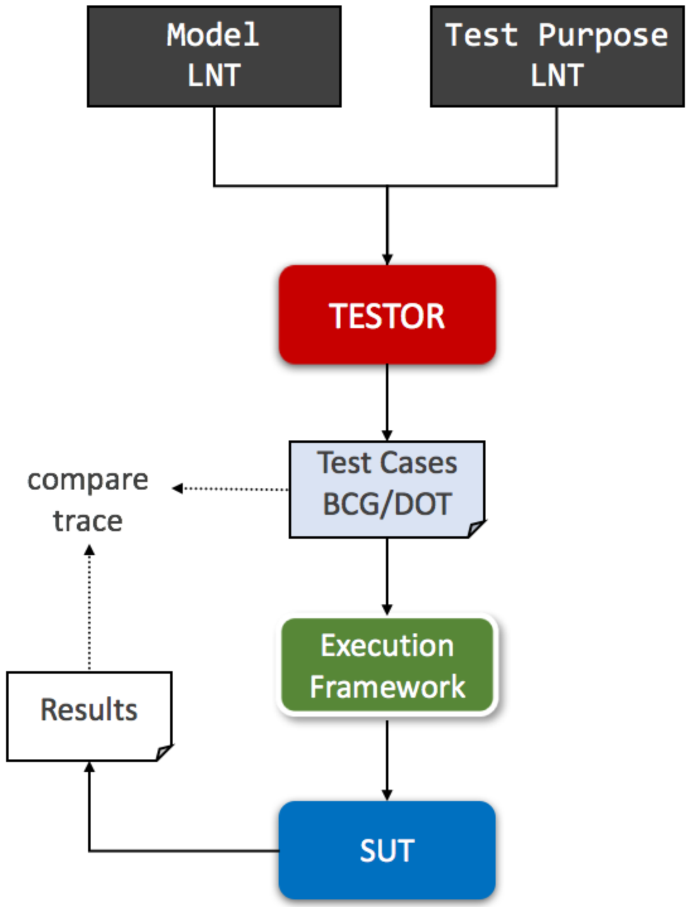
```
disrupt
     … content
by
     - - TLS 1.3 refuses renegotiation without a Hello Retry Request
    clientHello_c (?CH_p);
   alert := unexpected_message;
end disrupt
```

- "The client hello message can only arrive at the beginning of the handshake, or right after a hello retry request message."

Conformance Testing

- Model-based testing approach to compare the formal model of the TLS handshake with implementations.
- Extract test cases from the formal model.
- Run test cases on an implementation (SUT – *System Under Test*) and check whether the SUT conforms to the model.
- We used TESTOR [5], a recent tool for on-the-fly conformance test case generation guided by test purposes, developed on top of the CADP toolbox [6].
- The SUT in this validation process is an implementation of TLS 1.2.

Conformance Testing Overview



Test Purposes

- A test purpose aims to select a functionality to be tested by guiding the selection of test cases.
- Three test purposes corresponding to three requirements from the draft TLS 1.3 handshake specification:

  TP1. The protocol messages must be sent in the standard order (without the `HelloRetryRequest` message).

  TP2. The handshake must be aborted with an “unexpected message” alert, if there is a client renegotiation attempt.

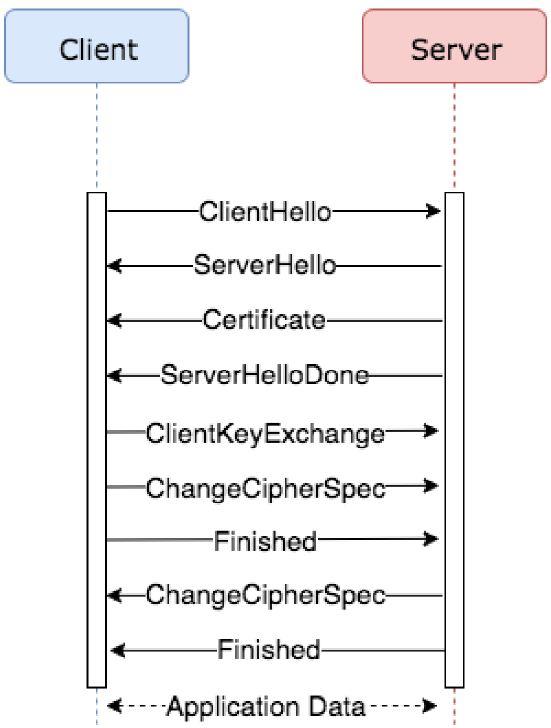
  TP3. The protocol messages are sent in the right order with an unexpected `CertificateRequest` (with a `HelloRetryRequest` message).

Test Cases

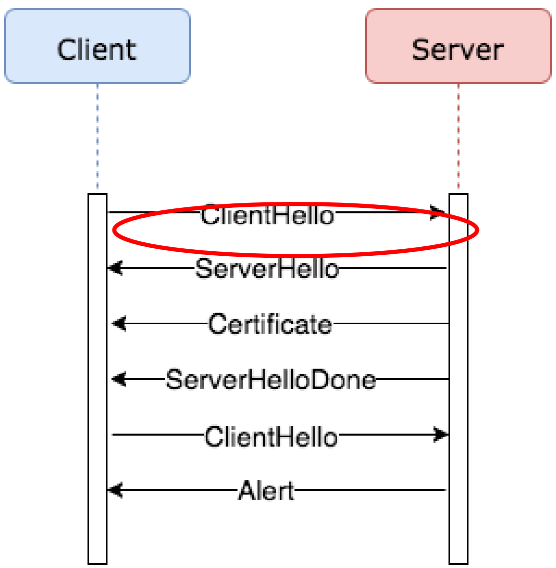
A test case (TC) is a sequence of interactions with the SUT.

TCi corresponds to one generated TC for a test purpose i.

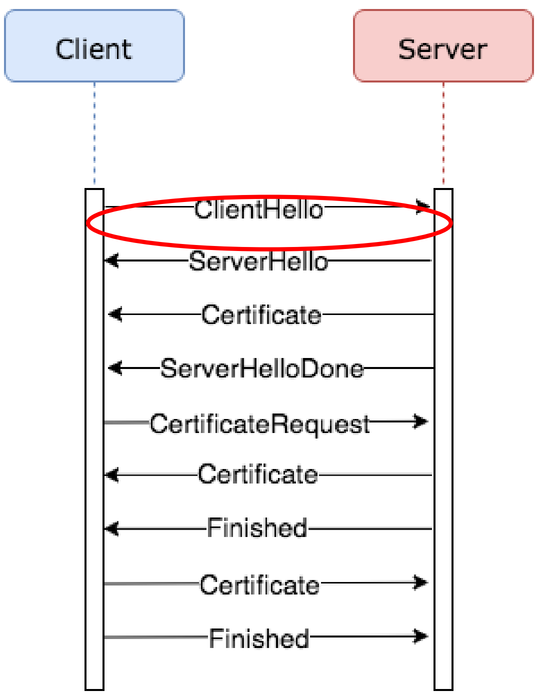
TC1. Standard TLS handshake



TC2. TLS handshake aborted by an Alert



TC3. TLS handshake with renegotiation



Test Execution

- Follow track of executed attack.
- Three possible verdicts:
  - *Pass:* Test purpose is reached.
  - *Fail*: The SUT is *not* conform to *M*.
  - *Inconclusive:* No indicative error encountered but the test purpose is not reached.

Test Execution Framework

- Emulate the interaction between client and server in a controlled and iterative way.
- Establish a connection to a TLS implementation with the execution framework and automatically test the SUT by following a formal specification from LNT.
- An adapted TLS-Attacker [7], an implementation for analyzing TLS libraries.
- Comprehends all TLS functionality according to v1.2 standard.

Test Execution Example (1/3)

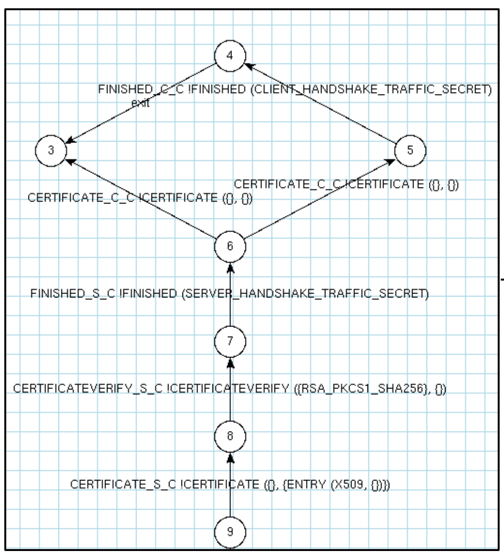
Pre	Action	Post
0	CLIENTHELLO	1
1	SERVERHELLO	2
2	CERTIFICATE_S	3
3	SERVERHELLODONE	4
4	CERTIFICATEREQUEST	5
5	CERTIFICATE_S	6
6	FINISHED_S	7
7	CERTIFICATE_C	8
8	FINISHED_C	9
9	exit	10

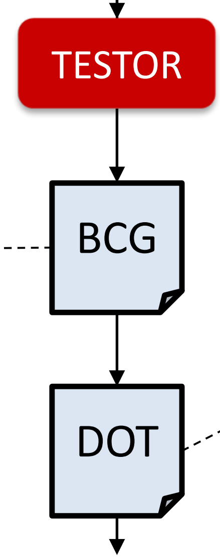
- The framework creates TLS messages on the fly according to the table, submits them against a SUT and reads its responses.
- Since no concrete values for the parameters of the messages are assigned, the tool generates default values automatically.

Test Execution Example (2/3)

```
process Client [clientHello_c: CH,
serverHello_c: SH,
certificate_c_c,
certificate_s_c: C,
certificateVerify_s_c: CV,
finished_c_c, finished_s_c: F,
alert_c: A] is ...
```

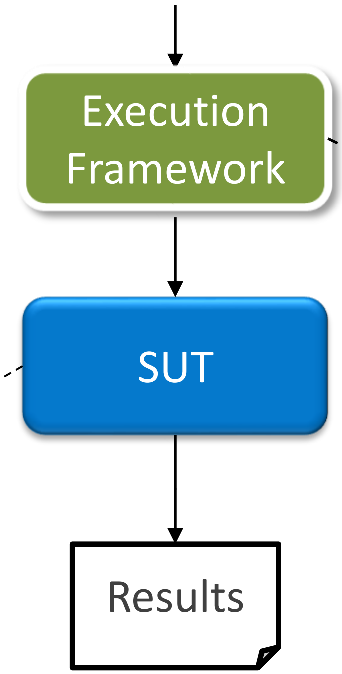
```
process ClientHello_TP [clienth: CH]
(is_hello_retry_request: bool,
in out CH_p: ClientHello,
HRR_p: HelloRetryRequest,
out alert: AlertType) is
...
```






```
digraph BCG {
size = "7, 10.5";
center = TRUE;
node [shape = circle];
0 [peripheries = 2];
0 -> 11 [label =
"CLIENTHELLO_C !CLIENTHELLO
(TLS12, 28BYTERAND, T_NULL,
{}, T_NULL, {EXTENSION
(SIGNATURE_ALGORITHMS,
SIGNATURESCHEMELIST
({RSA_PKCS1_SHA256,
RSA_PKCS1_SHA384,
RSA_PKCS1_SHA512,
ECDSA_SECP256R1_SHA256})),
EXTENSION
(SUPPORTED_VERSIONS,
SUPPORTEDVERSIONS
({TLS13}))})"];
```

Test Execution Example (3/3)



Pre	Action	Post
0	CLIENTHELLO	1
1	SERVERHELLO	2
2	CERTIFICATE_S	3
3	SERVERHELLODONE	4
4	CERTIFICATEREQUEST	5
5	CERTIFICATE_S	6
6	FINISHED_S	7
7	CERTIFICATE_C	8
8	FINISHED_C	9
9	exit	10

```
bin — openssl s_server -key key.pem -cert cert.pem -msg — 80×24
localhost:bin machine$ openssl s_server -key key.pem -cert cert.pem -msg
Using auto DH parameters
Using default temp ECDH parameters
ACCEPT
<<< TLS 1.2 Handshake [length 0067], ClientHello
    01 00 00 63 03 03 5a bb 80 72 21 d0 32 81 79 dd
    23 7f 00 41 1d a0 2d 25 9c db ff 48 0b 3c b7 41
    d1 1d ea 22 3e 1a 00 00 02 00 2f 01 00 00 38 00
    0a 00 0a 00 08 00 13 00 17 00 18 00 19 00 0b 00
    02 01 00 00 0d 00 20 00 1e 06 01 06 02 06 03 05
    01 05 02 05 03 04 01 04 02 04 03 03 01 03 02 03
    03 02 01 02 02 02 03
>>> TLS 1.2 Handshake [length 004a], ServerHello
    02 00 00 46 03 03 22 4b 5e d9 7b 5b 01 72 5c a5
    0a e2 63 a6 1b 24 bf 81 ac ed 98 2f 28 67 a3 ef
    78 2d 3a e4 4e e1 20 a5 be 4e c1 94 69 1b 15 16
    35 17 8b 31 3a e4 b4 07 92 83 11 ba 6e d8 12 2a
    02 26 ed ae 55 7c 7f 00 2f 00
>>> TLS 1.2 Handshake [length 03d7], Certificate
    0b 00 03 d3 00 03 d0 00 03 cd 30 82 03 c9 30 82
    02 b1 a0 03 02 01 02 02 09 00 d5 c8 c6 9e b9 17
    9b d4 30 0d 06 09 2a 86 48 86 f7 0d 01 01 0b 05
    00 30 7b 31 0b 30 09 06 03 55 04 06 13 02 44 45
    31 0f 30 0d 06 03 55 04 08 0c 06 42 65 72 6c 69
```

```
Action #1: CLIENT_HELLO
Action #2: SERVER_HELLO
Action #3: CERTIFICATE
Action #4: SERVER_HELLO_DONE
Action #5: CERTIFICATE_REQUEST
Action #6: ALERT
```

```
ALERT message:
  Level: FATAL
  Description:
UNEXPECTED_MESSAGE
```

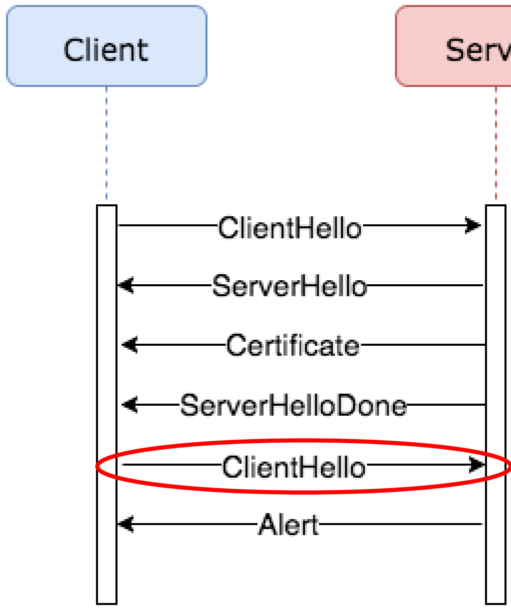
compare trace

verdict

Evaluation

- Framework: Automated execution.
- SUT: OpenSSL (TLS 1.2), https://www.openssl.org/.
- Applicability: Test a wide range of TLS implementations by only slightly manipulating the overall system.
- Test conformance to the formal LNT model of the TLS 1.3 handshake.

Evaluation: TC2 (1/2)



Obtained trace:

```
1: CLIENT_HELLO
2: SERVER_HELLO
3: CERTIFICATE
4: SERVER_HELLO_DONE
5: CLIENT_HELLO
6: ALERT
```

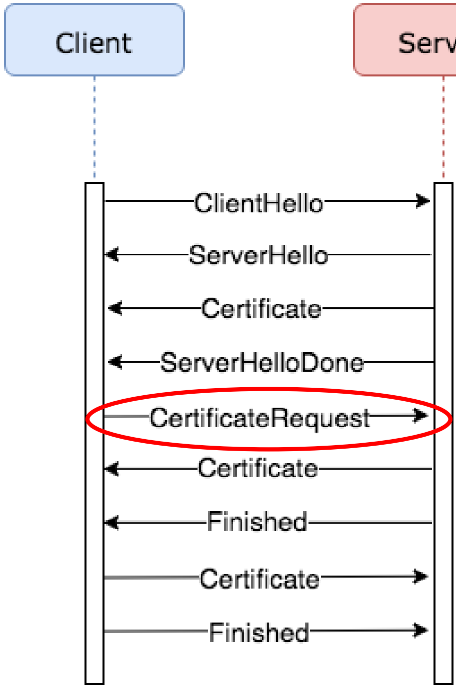
- The system responded as expected when being confronted with unexpected input.
- Thus, the behavior of the SUT is in conformance to the given TLS 1.3 Handshake LNT formal model.
- The test case is successful.

Evaluation: TC2 (2/2)

```
CLIENT_HELLO
  Handshake Message Length: 99
  Protocol Version: TLS12
  Client Unix Time: Wed Mar 28 13:45:54 CEST 2018
  Client Random:
  21 D0 32 81 79 DD 23 7F  00 41 1D A0 2D 25 9C DB
  FF 48 0B 3C B7 41 D1 1D  EA 22 3E 1A
  Session ID:
  Supported Cipher Suites: 00 2F
  Supported Compression Methods: 00
  Extensions:

SERVER_HELLO
  Handshake Message Length: 70
  Protocol Version: TLS12
  Server Unix Time: Sat Mar 26 08:33:45 CET 1988
  Server Random:
  7B 5B 01 72 5C A5 0A E2  63 A6 1B 24 BF 81 AC ED
  98 2F 28 67 A3 EF 78 2D  3A E4 4E E1
  Session ID:
  A5 BE 4E C1 94 69 1B 15  16 35 17 8B 31 3A E4 B4
  07 92 83 11 BA 6E D8 12  2A 02 26 ED AE 55 7C 7F
  Selected Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA
  Selected Compression Method: NULL
  Extensions:
```

```
CERTIFICATE_REQUEST
  Handshake Message Length: 18
  Certificate Types Count: 1
  Certificate Types: RSA_SIGN,
  Signature Hash Algorithms Length: 12
  Signature Hash Algorithms: SHA512-RSA,
SHA384-RSA, SHA256-RSA, SHA224-RSA, SHA1-RSA,
MD5-RSA,
  Distinguished Names Length: 0

                ...

ALERT
  Level: FATAL
  Description: UNEXPECTED_MESSAGE
```

Evaluation: TC3 (1/2)



Obtained trace:

```
1: CLIENT_HELLO
2: SERVER_HELLO
3: CERTIFICATE
4: SERVER_HELLO_DONE
5: CERTIFICATE_REQUEST
6: ALERT
```

```
routines:ACCEPT_SR_KEY_EXCH:unexpected message
```

- Output: The SUT does not reply to the request with the expected certificate.
- The server replies with an error and closes the connection.
- The `CertificateRequest` is not tolerated during this point of the handshake or a preceding concrete value causes the issue at this point.
- SUT does not behave in conformance to the model.

Related Work

- B. Beurdouche, A. Delignat-Lavaud, N. Kobeissi, A. Pironti, and K. Bhargavan. *FLEXTLS A Tool for Testing TLS Implementations*. WOOT'15, 2015.
- D. Kaloper-Meršinjak and H. Mehnert and A. Madhavapeddy and P. Sewell: *Not-Quite-So-Broken TLS: Lessons in Re-Engineering a Security Protocol Specification and Implementation*. USENIX Security 15, pp.223—238, 2015.

Conclusion

- Formal LNT model of the draft TLS Handshake protocol version 1.3.
- Validation of the model by using conformance testing.
- TLS implementations behave differently when being confronted with the same inputs [1].
- TLS implementations do not always follow the strict specification of the protocol.
- Conformance testing can help in order to detect the discrepancies.

Future Work

- Model:
  - Handle more extensions.
  - Implement optional messages (new session ticket, …).

- Validation:
  - Test TLS 1.3 implementations.
  - Specify known TLS attacks as test purposes.

References

[1] D. E. Simos, J. Bozic, F. Duan, B. Garn, K. Kleine, Y. Lei & F. Wotawa: *Testing TLS Using Combinatorial Methods and Execution Framework*. ICTSS'17 (10533), pp.162—177, 2017.

[2] *The Transport Layer Security (TLS) Protocol Version 1.3 draft-ietf-tls-tls13-24*, https://tools.ietf.org/html/draft-ietf-tls-tls13-24.

[3] The Transport Layer Security (TLS) Protocol Version 1.2, https://tools.ietf.org/rfc/rfc5246.

[4] *The DROWN Attack*, https://drownattack.com/.

[5] L. Marsso, R. Mateescu & W. Serwe: *TESTOR: A Modular Tool for On-the-Fly Conformance Test Case Generation*. TACAS'18 (10806), pp.211–228, 2018.

[6] H. Garavel, F. Lang, R. Mateescu & W. Serwe: *CADP 2011: a toolbox for the construction and analysis of distributed processes*. STTT 15(2), pp. 89–107, 2013.

[7] TLS-Attacker, https://github.com/RUB-NDS/TLS-Attacker.