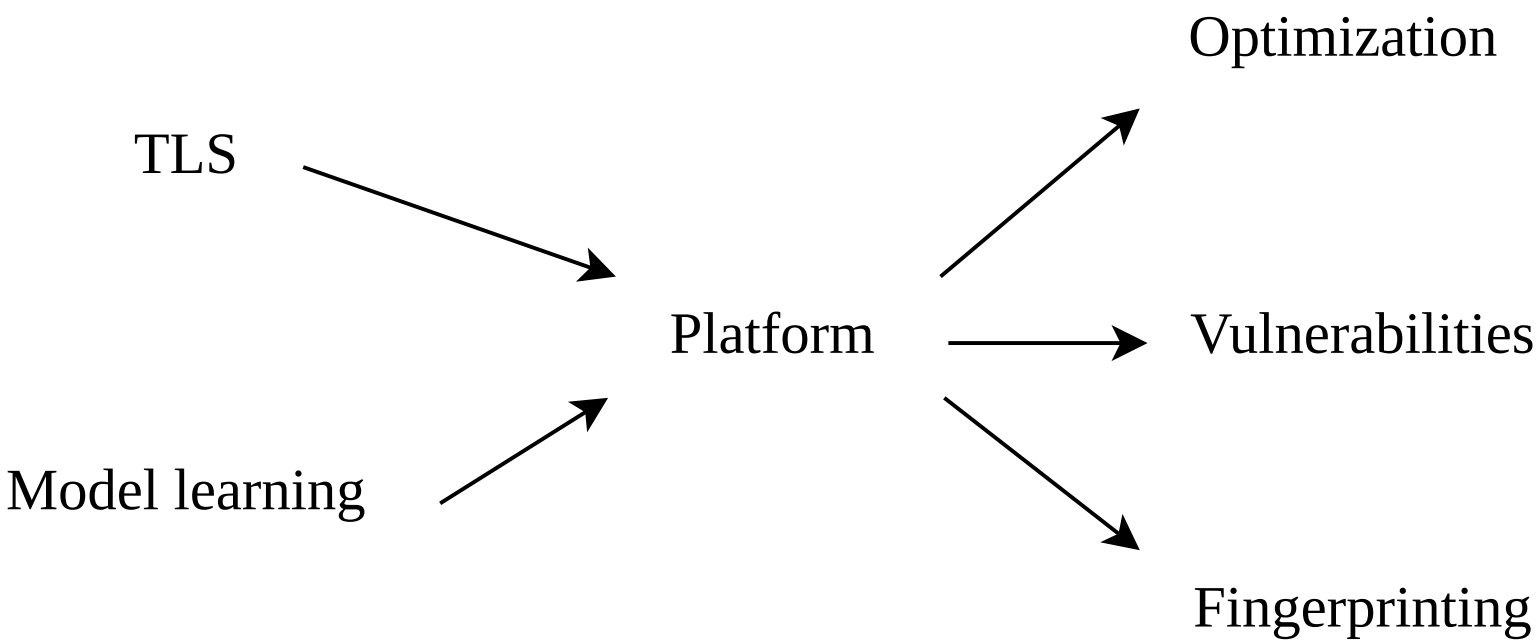
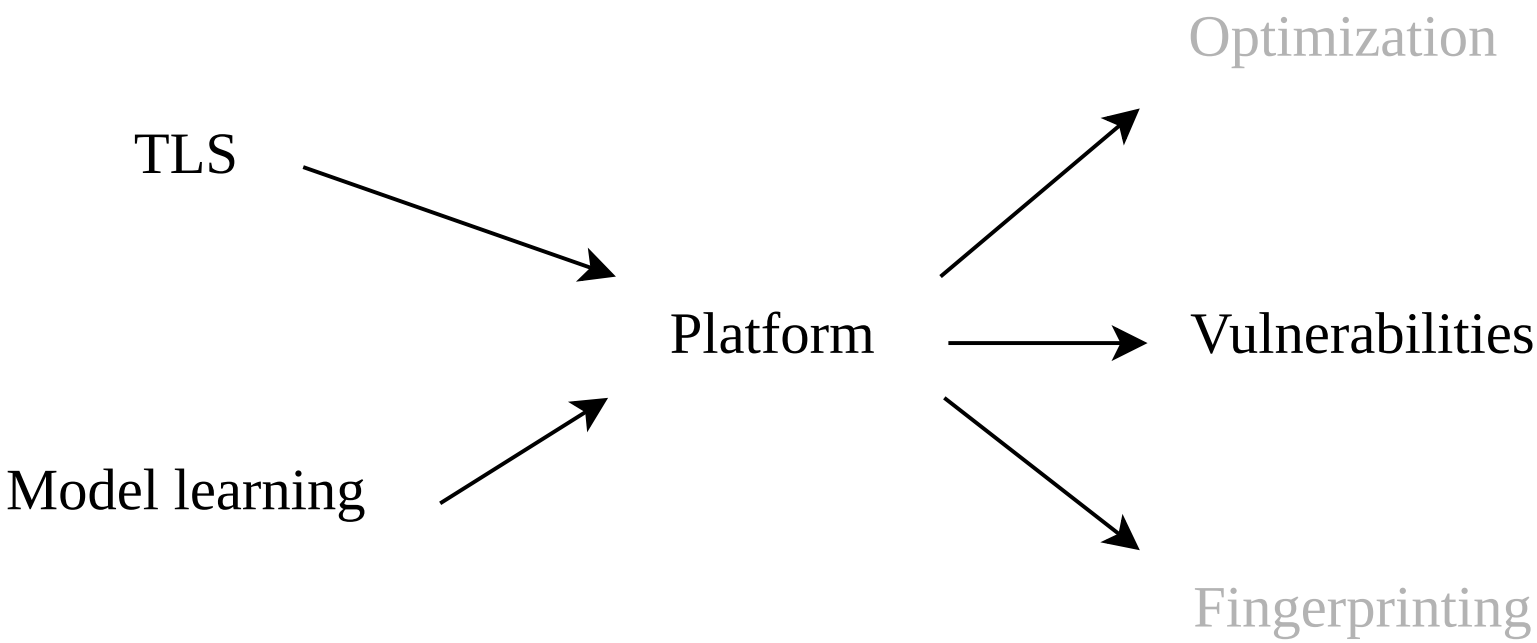
Towards a Systematic and Automatic Use of State Machine Inference to Uncover Security Flaws and Fingerprint TLS Stacks

Aina Toky Rasoamanana, Olivier Levillain and Hervé Debar

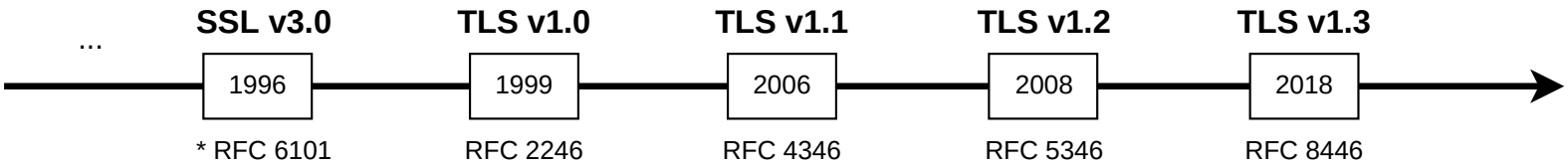
Télécom SudParis – Institut Polytechnique de Paris

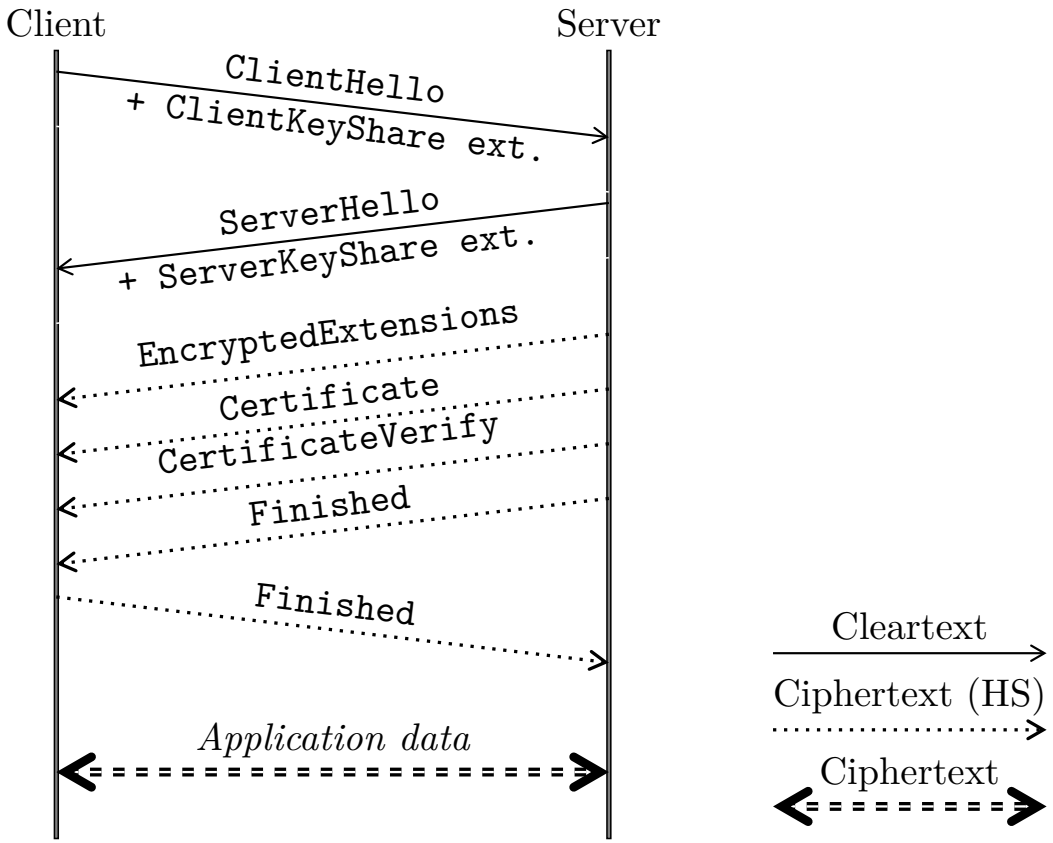
September 28, 2022


TLS
Model learning
Platform
Optimization
Vulnerabilities
Fingerprinting


Optimization
TLS
Platform
Vulnerabilities
Model learning
Fingerprinting

Short introduction to TLS





Related Work – TLS State Machines

- Known state machines attacks:
  - CVE-2014-0224: `EarlyCCS`
  - CVE-2015-0204: `FREAK`
  - CVE-2015-0205: `SkipVerify`
- Other known attacks:
  - Bleichenbacher attack (`ROBOT`)
  - CVE-2014-6321: WinShock
- Different methods to analyze TLS implementations:
  - `TLS-Attacker` [Som16]
  - `FlexTLS` [BDLK+15]
  - Model learning [dRP15]

Our Contributions

- Systematic and reproducible
- Large-scale experimentation
  - around 500 stacks/versions
- Efficiency
  - up to 20 times faster than `statelearner` and unoptimized `pylstar`
- Open sourced tools[1]

[1] `pylstar-tls`: `https://gitlab.com/gaspians/pylstar-tls.git`

Model Learning with L*

Goal: infer a deterministic Mealy machine using abstract messages

1. Membership queries

   - SUT = System Under Test
   - update an observation table
   - result: a conjectured state machine

2. Equivalence queries
   - input: conjectured state machine
   - output: true or counter-example
   - Approximation with membership queries

Model Learning with L*

Goal: infer a deterministic Mealy machine using abstract messages

1. Membership queries

   - SUT = System Under Test
   - update an observation table
   - result: a conjectured state machine

2. Equivalence queries
   - input: conjectured state machine
   - output: true or counter-example
   - Approximation with membership queries

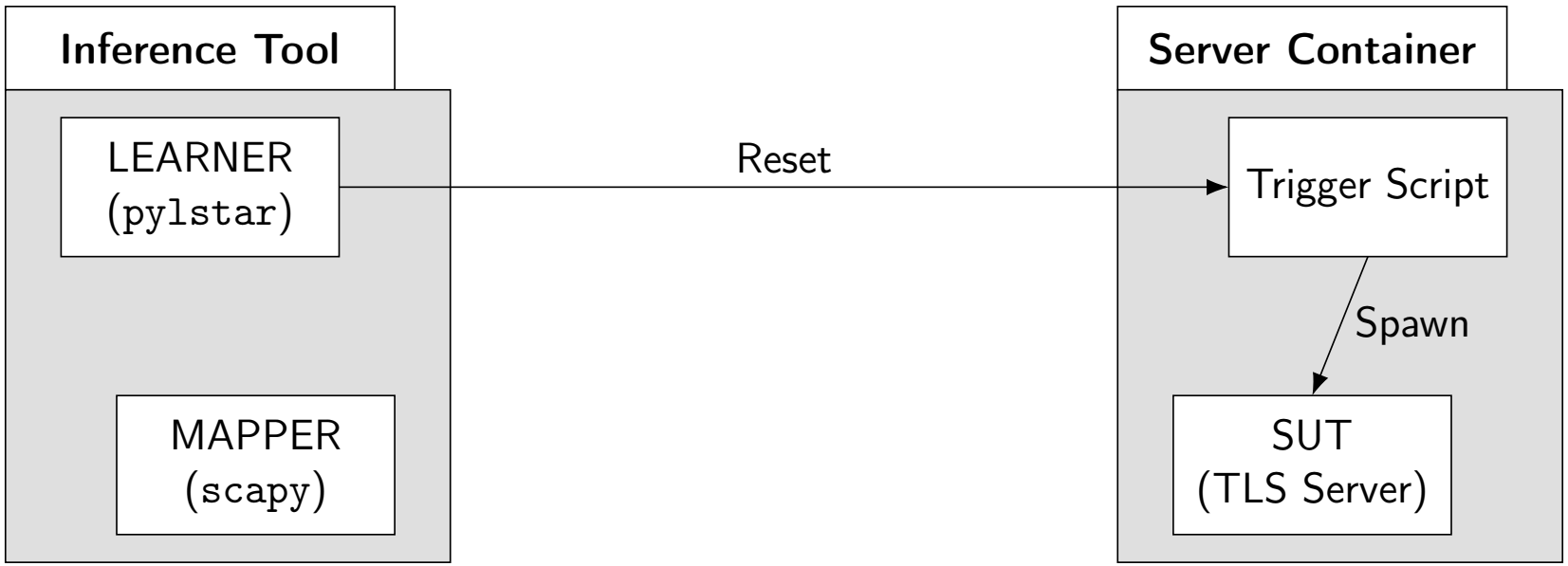
Model Learning – TLS

Model Learning – TLS

Trying Sequence *ClientHello*, *ClientKeyExchange* with a TLS 1.2 server

Model Learning – TLS

Trying Sequence *ClientHello*, *ClientKeyExchange* with a TLS 1.2 server



Model Learning – TLS

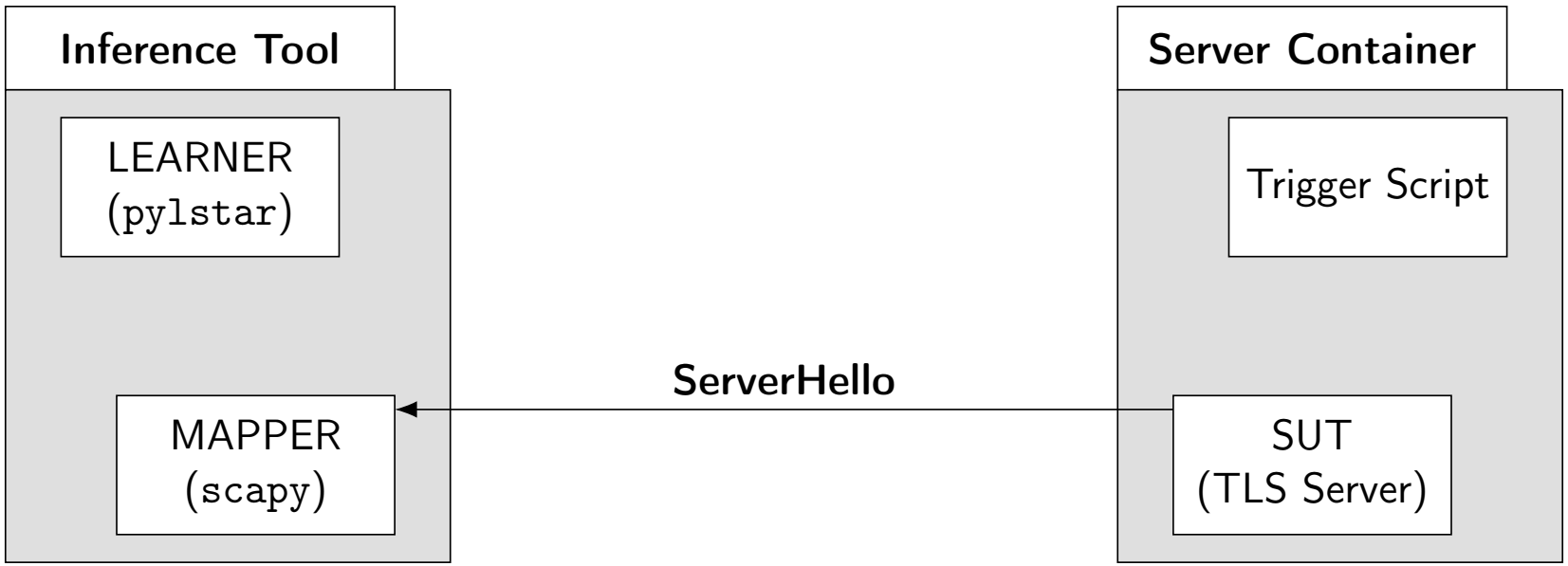
Trying Sequence *ClientHello*, *ClientKeyExchange* with a TLS 1.2 server

Model Learning – TLS

Trying Sequence *ClientHello*, *ClientKeyExchange* with a TLS 1.2 server

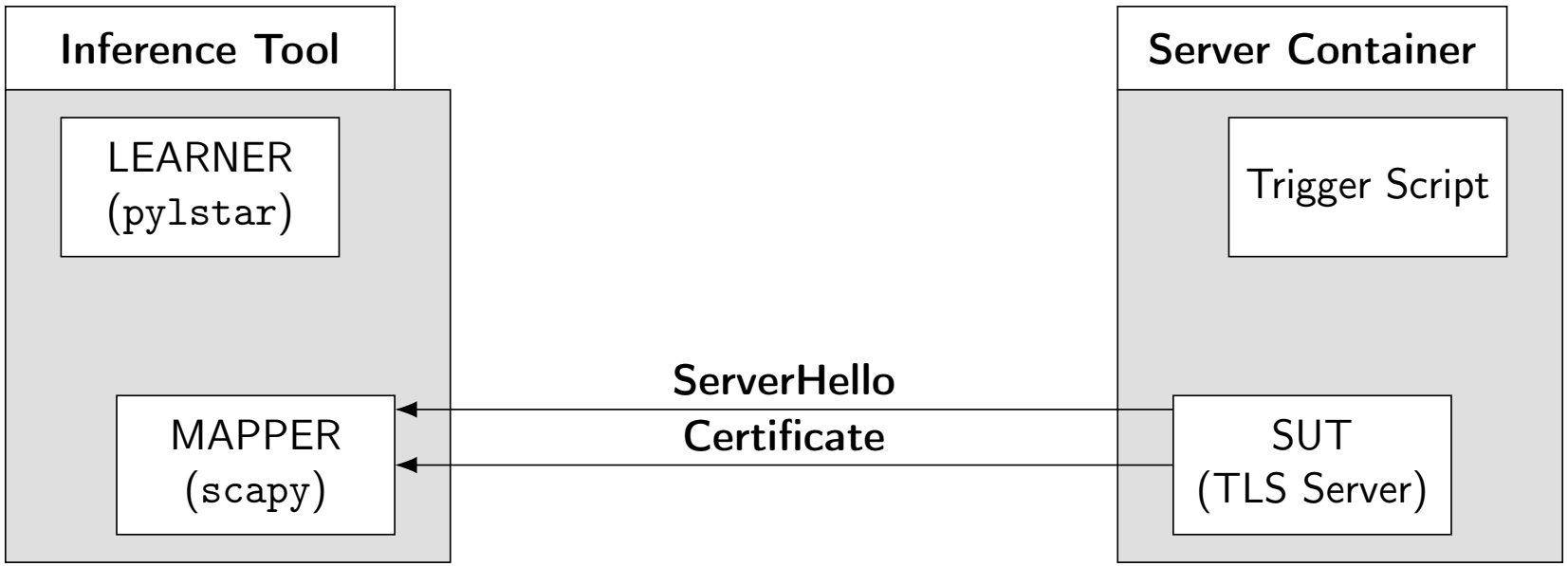
Model Learning – TLS

Trying Sequence *ClientHello*, *ClientKeyExchange* with a TLS 1.2 server



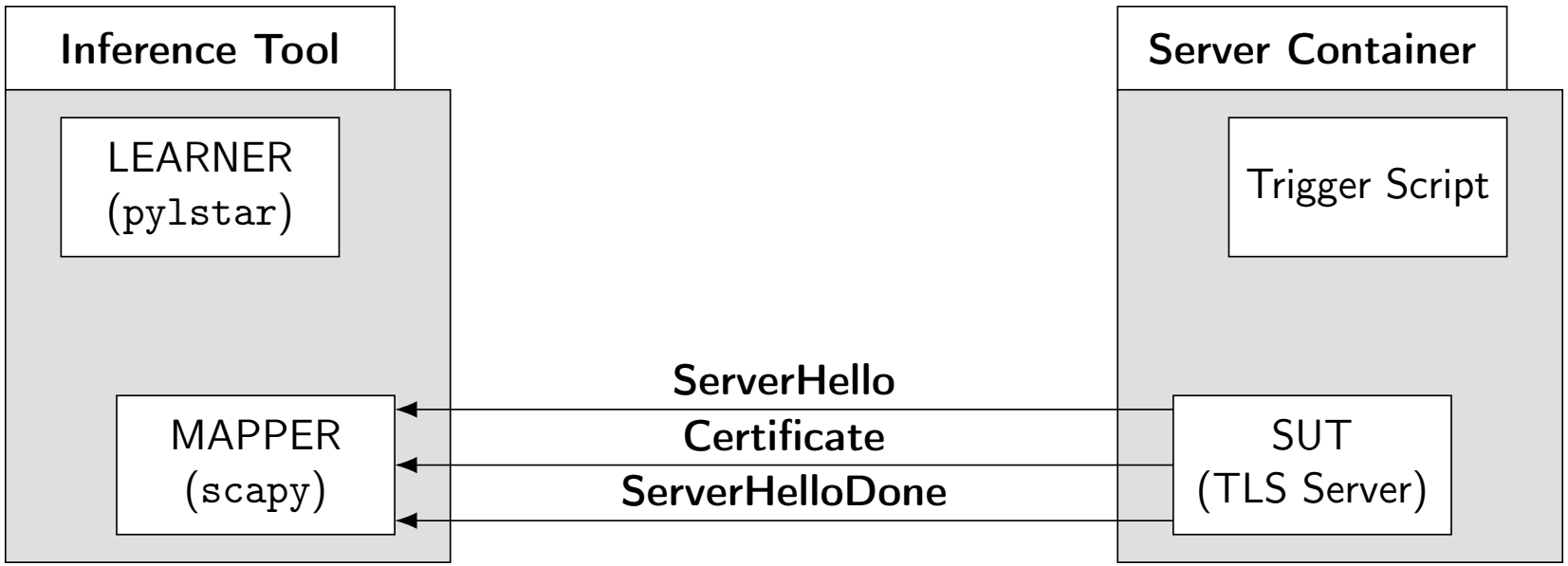
Model Learning – TLS

Trying Sequence *ClientHello*, *ClientKeyExchange* with a TLS 1.2 server



Model Learning – TLS

Trying Sequence *ClientHello*, *ClientKeyExchange* with a TLS 1.2 server



Model Learning – TLS

Trying Sequence *ClientHello*, *ClientKeyExchange* with a TLS 1.2 server

Model Learning – TLS

Trying Sequence $\mathcal{ClientHello}$, $\mathcal{ClientKeyExchange}$ with a TLS 1.2 server

$$\mathcal{ClientHello} \implies \mathcal{ServerHello} + \mathcal{Certificate} + \mathcal{ServerHelloDone}$$

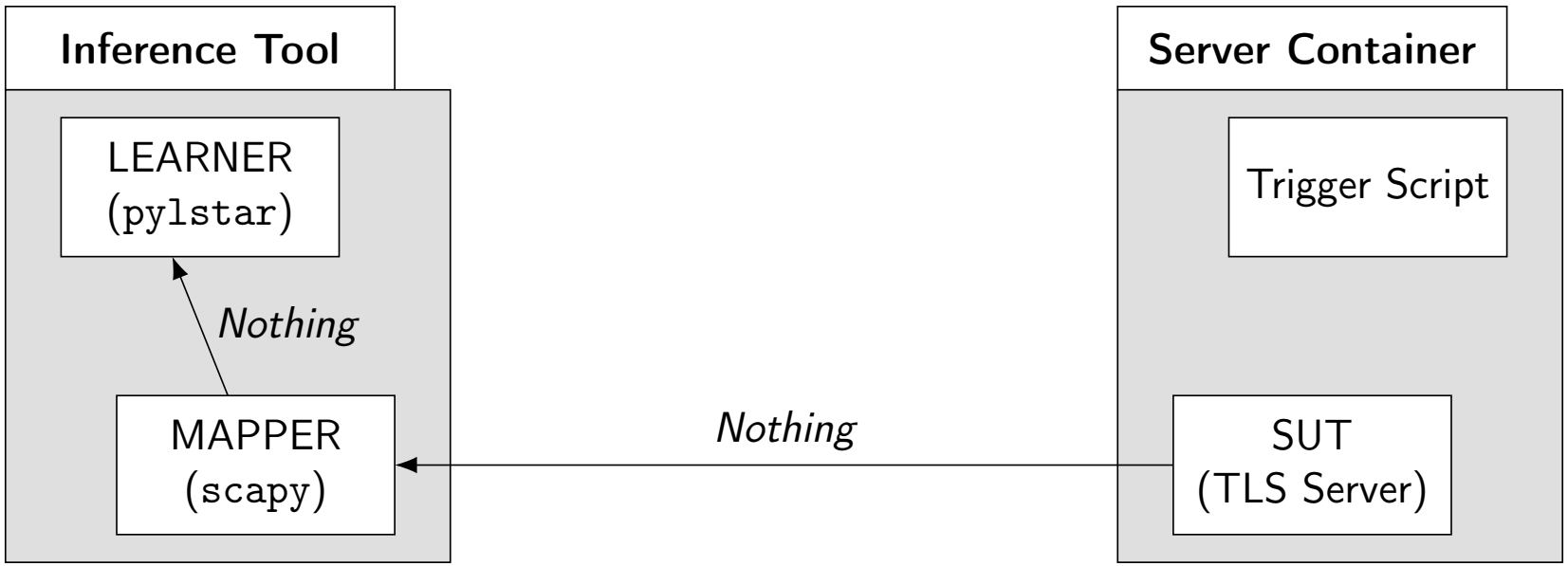
Model Learning – TLS

Trying Sequence *ClientHello*, *ClientKeyExchange* with a TLS 1.2 server

ClientHello $\Longrightarrow$ *ServerHello* + *Certificate* + *ServerHelloDone*

Model Learning – TLS

Trying Sequence $\mathcal{ClientHello}$, $\mathcal{ClientKeyExchange}$ with a TLS 1.2 server



$$\mathcal{ClientHello} \implies \mathcal{ServerHello} + \mathcal{Certificate} + \mathcal{ServerHelloDone}$$

Model Learning – TLS

Trying Sequence $\mathcal{ClientHello}$, $\mathcal{ClientKeyExchange}$ with a TLS 1.2 server

$$\mathcal{ClientHello} \Longrightarrow \mathcal{ServerHello} + \mathcal{Certificate} + \mathcal{ServerHelloDone}$$
$$\mathcal{ClientKeyExchange} \Longrightarrow \textit{Nothing}$$

Challenges on Model Learning

Assumptions

- Connection independance
- Deterministic behavior of the SUT

Challenges to find a balance between efficiency and accuracy

- find a relevant approximation for the equivalence query
- choose an appropriate timeout value
  - avoid non-deterministic behaviors
- develop a robust and flexible `MAPPER`
  - e.g. *ClientHello*, *Finished*, *ClientKeyExchange*
- ensure the SUT is reset properly before each query

Scenarios

scenario := (`role, TLS version, context`), where:

- `role`: client/server
- `version`: `TLS 1.0, 1.1, 1.2, 1.3`
- `context`
  - normal
  - mutual authentication
  - broken authentication
    - `EmptyCertificate`
    - `CertificateVerify` invalid
    - Untrusted `Certificate`
  - `FREAK`
  - Bleichenbacher oracle
- e.g. `(server, 1.3, mutual_authentication)`

TLS Test Bed

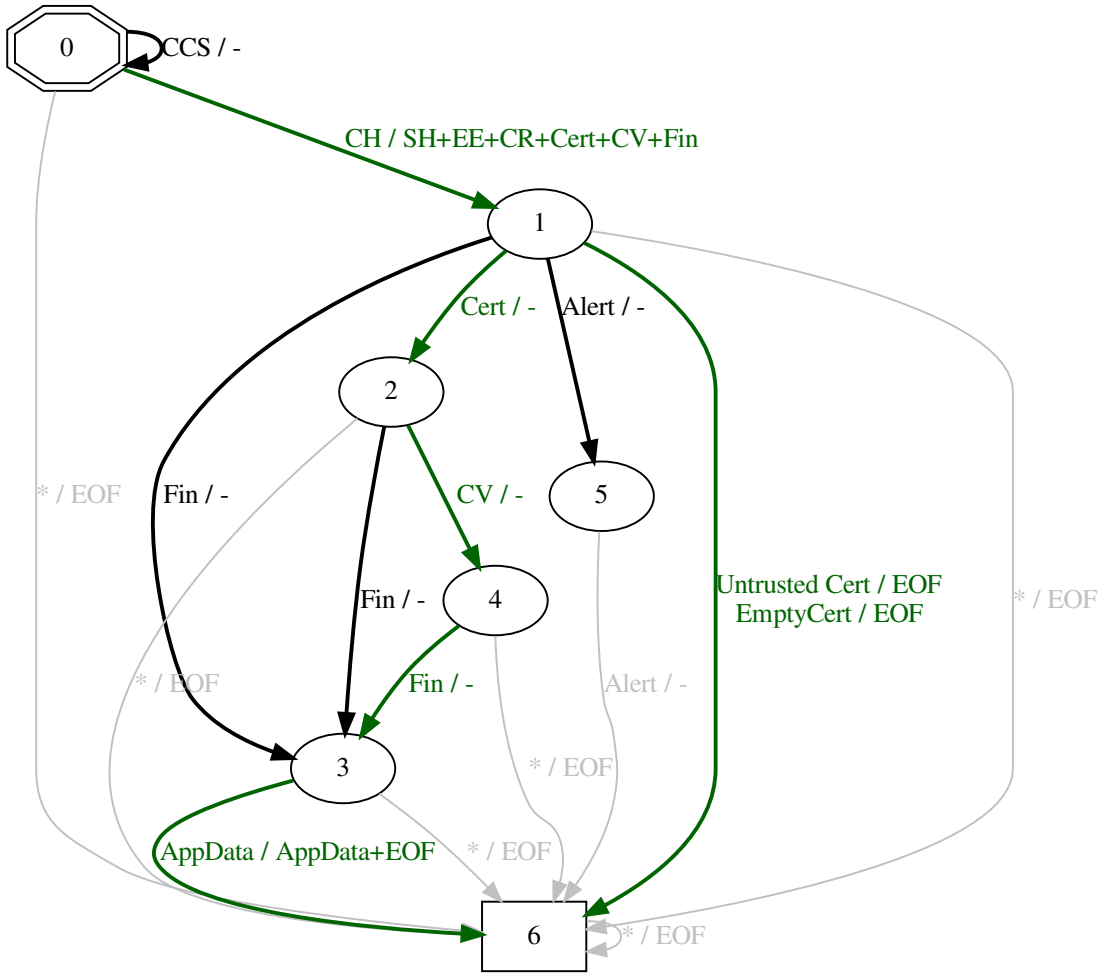
Stack Name	Versions	Client	Server
OpenSSL	0.9.8m – 3.0.2	✓	✓
curl + OpenSSL	1.1.1a – 3.0.2	✓	
GnuTLS	3.6.16 – 3.7.2	✓	✓
curl + GnuTLS			
mbedtls	1.3.10 – 3.0.0p1	✓	✓
wolfSSL	3.12.0 – 5.1.1	✓	✓
curl + wolfSSL			
matrixssl	3.7.2, 4.0.0 – 4.3.0		✓
NSS	3.15 – 3.78	✓	✓
erlang	20.0, 24.0.3 – 24.2.1		✓
fizz	2021.02 – 2021.06	✓	

Automatic and Efficient Bugs Detection

1. Learn state machines for a given scenario
   - Complex state machines
     - ★ we have around 4,000 state machines
   - Need rigorous method to analyze them
2. Analyze results in 2-steps process

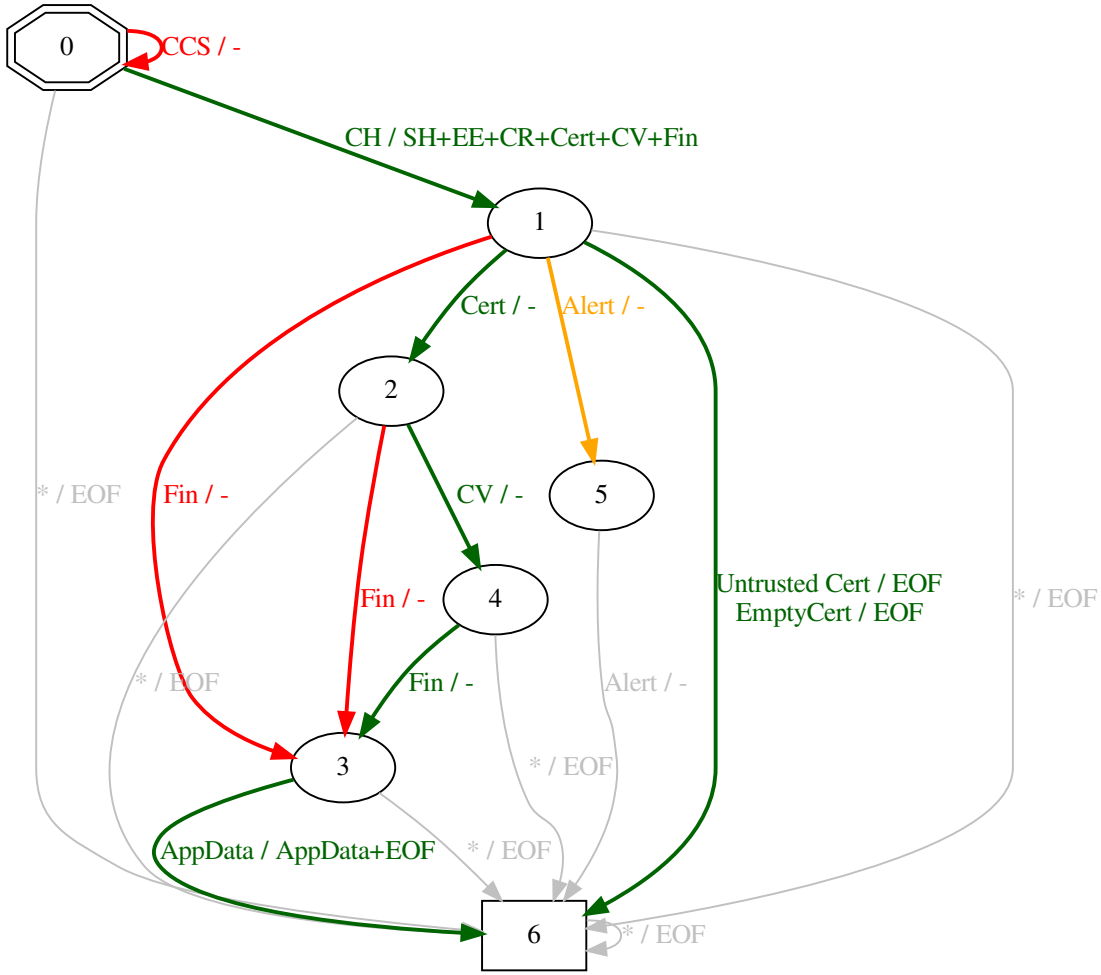
Automatic and Efficient Bugs Detection

- Color in green the happy paths
- Merge and color in gray error transitions leading to the sink state



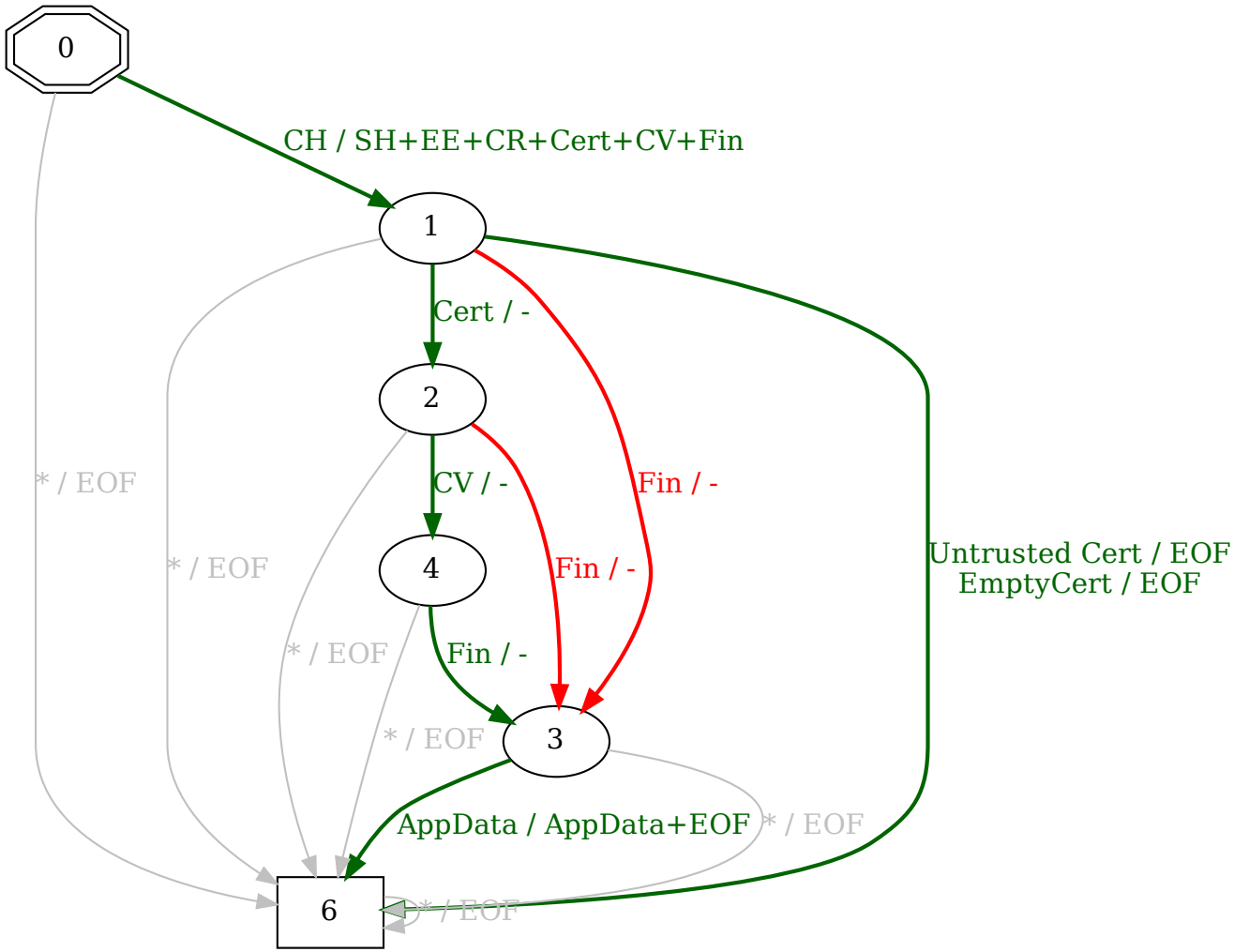
Automatic and Efficient Bugs Detection

- Remaining transitions are either vulnerabilities or RFC violations
  - Vulnerabilities may require confirmation
  - RFC violations may require further study



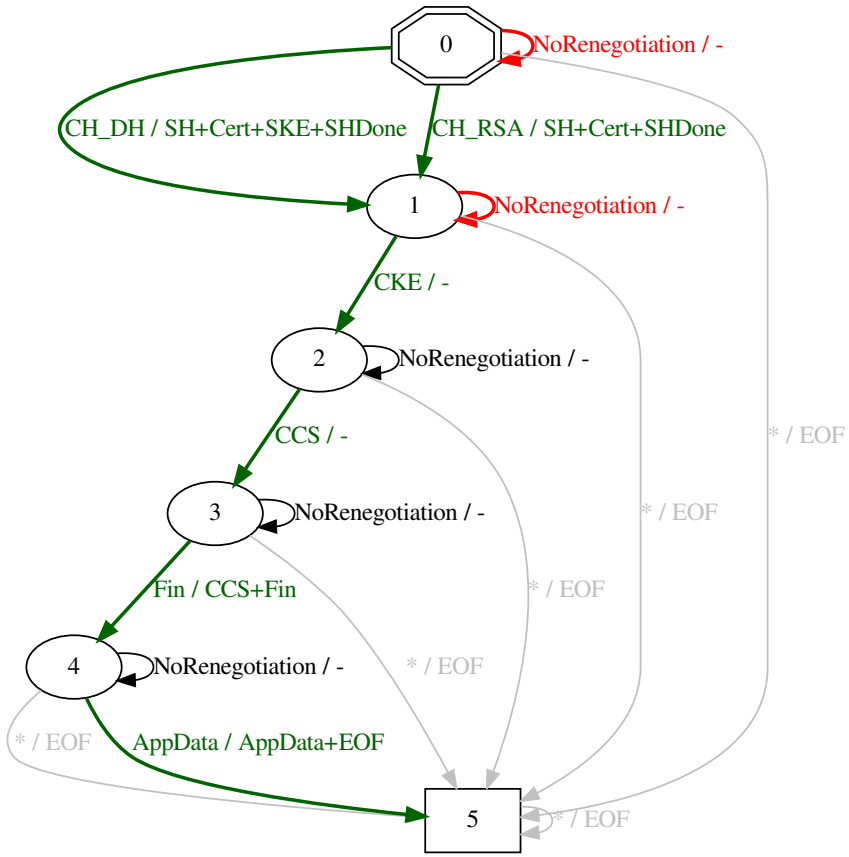
Authentication Bypasses – CVE-2022-25640

- Scenario: `TLS 1.3` server with mutual authentication
- `wolfSSL` before 5.2.0



Loops and DoS Attack

- Loops before encryption/authentication
- Allow attacker to lead DoS attack with very few resources (e.g: CVE-2020-12457)
- e.g. scenario: `TLS` 1.2 server normal (`NSS` stack)



New and Reproduced Bugs

- Authentication bypasses
  - CVE-2022-25640 *New*
  - CVE-2022-25638 *New*
  - CVE-2021-3336 *New*
  - CVE-2020-24613
  - CVE-2015-0204 (`FREAK`)
  - CVE-2014-0224 (`EarlyCCS`)
- DoS attack – Loops
  - CVE-2022-25639 *New*
  - Loops with `TLS` 1.0 to 1.2 NSS server (ongoing discussion) *New*
  - Loops with `TLS` 1.3 fizz client (benign) *New*
  - CVE-2020-0224
- Bleichenbacher oracle (`ROBOT`)
  - CVE-2017-1000385
  - CVE-2017-13099
  - CVE-2016-6883

Conclusion and Future Work

- An extensive work on TLS state machine inference
  - several scenarios considered
  - bug finding automation
  - around 500 stacks/versions
  - optimization of the $L^\star$ approach
  - state-machine-based fingerprinting
- Future work
  - Consider all possible features affecting state machines
    - ★ TLS 1.3 middlebox compatibility
    - ★ renegotiation
  - Extension of the approach to other protocols

References


Benjamin Beurdouche, Antoine Delignat-Lavaud, Nadim Kobeissi, Alfredo Pironti et Karthikeyan Bhargavan :
FLEXTLS: A Tool for Testing TLS Implementations.
In 9th USENIX Workshop on Offensive Technologies (WOOT 15), 2015.


Joeri de Ruiter et Erik Poll :
Protocol State Fuzzing of TLS Implementations.
In 24th USENIX Security Symposium (USENIX Security 15), pages 193–206, 2015.


Juraj Somorovsky :
Systematic Fuzzing and Testing of TLS Libraries.
In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pages 1492–1504, 2016.

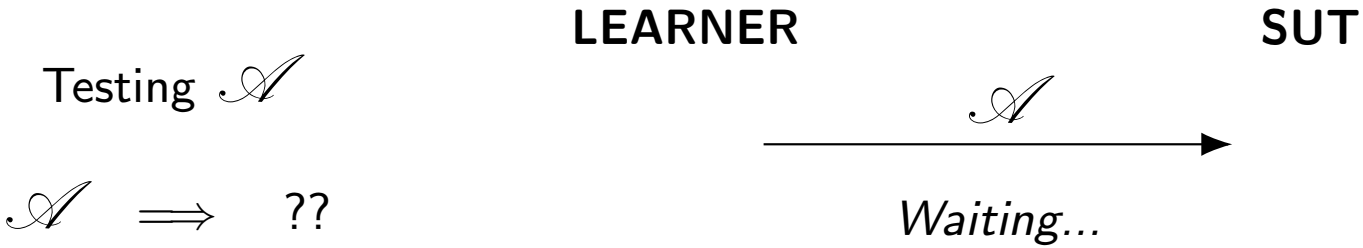
Backup slides

Optimization

- Exploiting determinism – use prior knowledge to avoid useless waiting

Optimization

- Exploiting determinism – use prior knowledge to avoid useless waiting

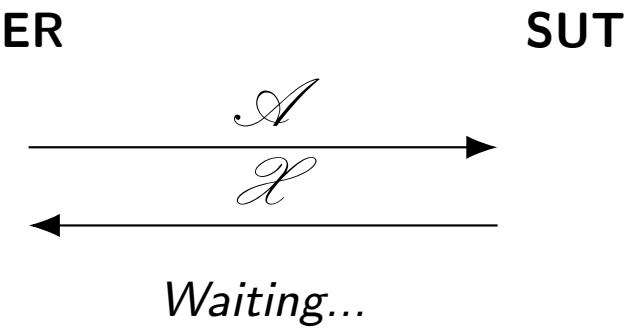


Optimization

- Exploiting determinism – use prior knowledge to avoid useless waiting

Testing $\mathscr{A}$

$\mathscr{A} \implies$??

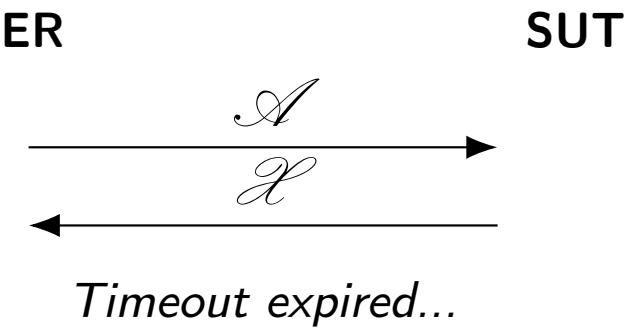


Optimization

- Exploiting determinism – use prior knowledge to avoid useless waiting

Testing $\mathscr{A}$

$\mathscr{A} \implies$??

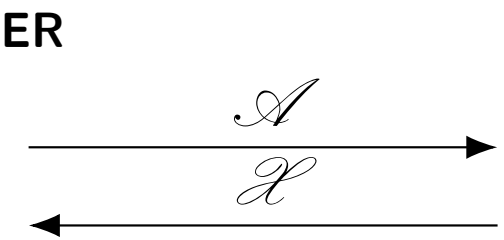


Optimization

- Exploiting determinism – use prior knowledge to avoid useless waiting

Testing $\mathscr{A}$

$\mathscr{A} \implies \mathscr{X}$



Optimization

- Exploiting determinism – use prior knowledge to avoid useless waiting

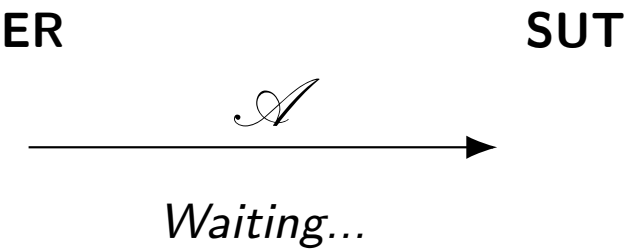
Optimization

- Exploiting determinism – use prior knowledge to avoid useless waiting

Testing $\mathscr{A}$, $\mathscr{B}$

$\mathscr{A} \implies \mathscr{X}$

$\mathscr{B} \implies$??



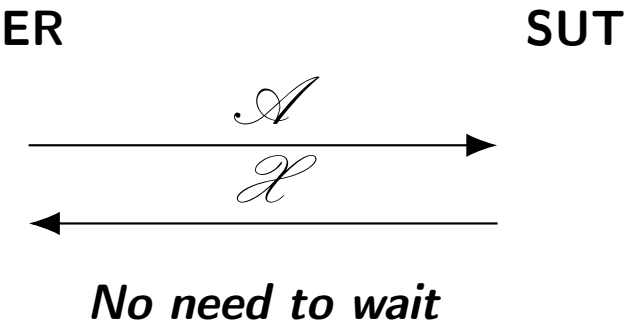
Optimization

- Exploiting determinism – use prior knowledge to avoid useless waiting

Testing $\mathscr{A}$, $\mathscr{B}$

$\mathscr{A} \implies \mathscr{X}$

$\mathscr{B} \implies$??



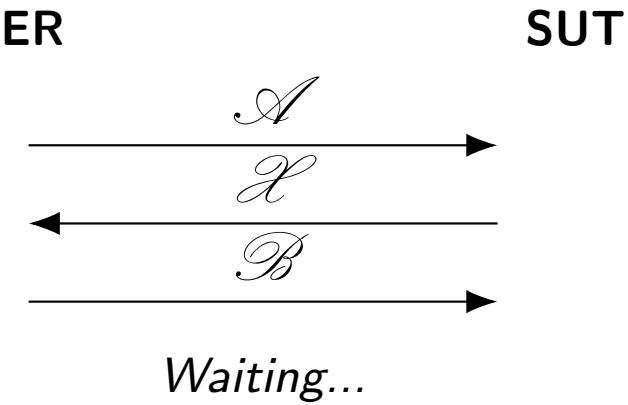
Optimization

- Exploiting determinism – use prior knowledge to avoid useless waiting

Testing $\mathscr{A}$, $\mathscr{B}$

$\mathscr{A} \implies \mathscr{X}$

$\mathscr{B} \implies$??



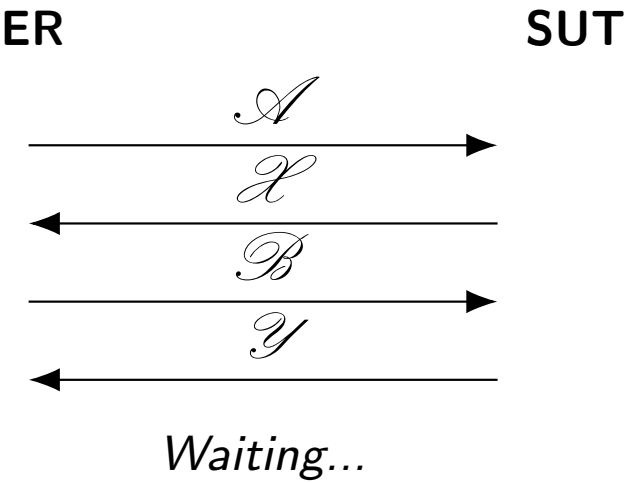
Optimization

- Exploiting determinism – use prior knowledge to avoid useless waiting

Testing $\mathscr{A}$, $\mathscr{B}$

$\mathscr{A} \implies \mathscr{X}$

$\mathscr{B} \implies$??

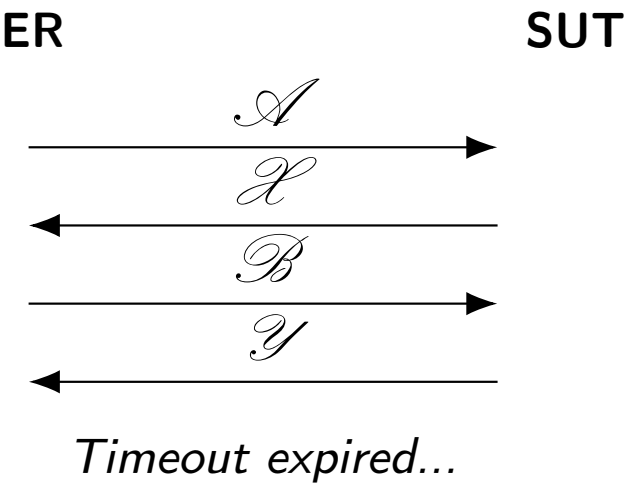


Optimization

- Exploiting determinism – use prior knowledge to avoid useless waiting

Testing $\mathscr{A}$, $\mathscr{B}$

$$\begin{array}{lll} \mathscr{A} & \Longrightarrow & \mathscr{X} \\ \mathscr{B} & \Longrightarrow & \mathscr{Y} \end{array}$$



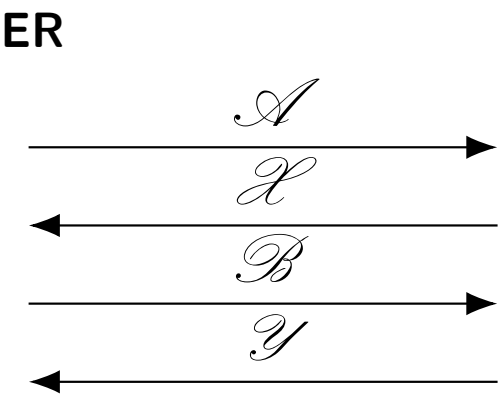
Optimization

- Exploiting determinism – use prior knowledge to avoid useless waiting

Testing $\mathscr{A}$, $\mathscr{B}$

$\mathscr{A} \implies \mathscr{X}$

$\mathscr{B} \implies$



- `EOF` (= *connection closed*) is final

	No `EOF` optim.	**`EOF` optimization**
No anticipation	1,885 s (100 %)	1,598 s (85 %)
Skip timeouts on empty responses	1,081 s (57 %)	862 s (46 %)
Skip timeouts on all known responses	128 s (7 %)	77 s (4 %)

TLS Fingerprinting

- Difference in state machines of different stacks
  - Bugs and vulnerabilities
  - Different alert messages
- Compute the *distinguishing sequences* for a given scenario:
  - Find sequences to separate 2 state machines and so on
  - Fingerprints are the output corresponding to these sequences

Fingerprinting – TLS 1.3 servers

- Identify the stack/version classes of a given SUT

Stacks	Versions	Nb states	Comment
erlang	24.0.3 – 24.2.1	9	Separating these 13 classes only requires sending 8 distinguishing sequences
GnuTLS	3.6.16 – 3.7.2	4	
matrixssl	4.0.0 – 4.1.0	4	
	4.2.1 – 4.3.0	6	
NSS	3.39 – 3.40	4	
	3.41 – 3.78	4	
OpenSSL	1.1.1a – 1.1.1n	4	
	3.0.0 – 3.0.2	4	
wolfSSL	3.15.5 – 4.0.0	7	
	4.1.0 – 4.6.0	7	
	4.7.0 – 4.8.1	7	
	5.0.0 – 5.1.1	7	
	5.2.0	6	