ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE
ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL
ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

USENIX Security '25 Artifact Appendix: Towards Internet-Based State Learning of TLS State Machines

Marcel Maehren[1], Nurullah Erinola[1], Robert Merget[2], Jörg Schwenk[1], and Juraj Somorovsky[3]

[1]Ruhr University Bochum
[2]Technology Innovation Institute
[3]Paderborn University

A Artifact Appendix

A.1 Abstract

In our main paper, we apply state machine learning to TLS deployments on the Internet and introduce techniques to address challenges arising from the uncontrolled environments. We further propose a novel methodology for an automated analysis of learned TLS state machines that allows us to identify vulnerabilities and non-compliant message flows. In a large-scale study, we collected 1304 state machines of real-world TLS hosts, uncovering several uncritical deviations, but also domains vulnerable to padding oracle attacks, and domains violating the session transcript integrity, potentially enabling MitM attacks.

This artifact appendix aims to show the functionality of the state learner we developed to conduct our study. The experiments involve the automated extraction of state machines from two versions of OpenSSL via Docker. First, we consider OpenSSL 3.4.0 a modern version exhibiting no state machine issues. Subsequently, we extract a state machine for OpenSSL 1.0.1j exhibiting an issue similar to the NetScaler transcript integrity vulnerability we discuss in our paper. For both versions, the respective experiment runs our automated analysis to test for issues. Finally, in a third experiment, we consider the analysis of anonymized state machines contained in our dataset to illustrate selected issues we observed in our evaluations.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

We are not aware of any security, privacy, or ethical concerns arising from running the experiments described below.

A.2.2 How to access

We provide our dataset via Zenodo.[1]

A.2.3 Hardware dependencies

For running the experiments, please ensure that the Java Virtual Machine can allocate at least 8GB of RAM.

A.2.4 Software dependencies

The experiments expect the ability to run bash scripts, and basic Linux CLI tools (wget/curl, awk, git, unzip, and grep). As part of the setup, Docker must be installed.

A.2.5 Benchmarks

Running the tool for experiment `E3` requires about 5 hours of computational time on a standard consumer grade desktop PC. Free disk space of 5 GB is required to build Docker images and for the dataset from our Zenodo repository.

A.3 Set-up

A.3.1 Installation

Please follow the instructions at https://docs.docker.com/get-started/get-docker/ to install Docker on your system. Please configure the permissions so that the user executing the scripts provided as part of this artifact is allowed to `docker build` and `docker run`. Subsequently, download and unzip the `artifact_experiments.zip` file from the Zenodo repository:

```
https://doi.org/10.5281/zenodo.15520932
```

Run the `setup.sh` script from the root directory of the artifact experiment directory. This script performs the following steps:

1. Downloads our source code from Zenodo.
2. Builds three Docker images: First, the script builds two images of OpenSSL 3.4.0 and OpenSSL 1.0.1j combined with our state learner tool. As part of this process, the source code fetched from Zenodo will be built using

[1] https://doi.org/10.5281/zenodo.15520932

Maven. Subsequently, the script builds a Docker image for our CLI tool (State Machine Analysis Tool).

3. Downloads our dataset from Zenodo.
4. Extracts the dataset to `experiment_dataset/`.
5. Creates output directories for experiments.
6. Executes a basic test to verify the setup (see Section A.3.2).
7. Asks if computational steps from E1 and E2 expected to take 5 hours and 10 minutes should be run immediately.

Note that you can skip the last step and perform the computational steps as part of the experiments later on.

A.3.2 Basic Test

The basic test is automatically executed as part of `setup.sh`. It runs the OpenSSL 3.4.0 container with minimal parameters to quickly verify functionality. The test:

1. Runs our tool to extract a basic state machine using Docker. As part of this process, the Docker container will open tmux with two panes showing OpenSSL on the left and our tool on the right. This test is expect to finish within 5 minutes. Results will be written to: `experiment_outputs/Basic_Test/alphabet-1/`
2. Verifies the creation of three output files:
   - `OpenSSL3.4.0.xml` - The state machine in XML format
   - `OpenSSL3.4.0_short.pdf` - A visualization of the state machine
   - `OpenSSL3.4.0_short.dot` - Analysis details in DOT format
3. Checks that the DOT file indicates that the analyses runs correctly

Expected output includes messages confirming successful Docker builds, dataset extraction, and verification that all expected files were created with correct content.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): We claim to provide a tool to conduct state learning for TLS implementations. The tool iterates over different alphabets to yield increasingly detailed models of the state machine of an implementation (cf. Section 5 of our main paper). Throughout the execution, we employ a cache which achieves a high coverage rate (>=97% for the individual alphabets).

(C2): We claim to provide an automated analysis for the detection of state machine issues (cf. Section 5.6 of our main paper).

(C3): We claim to provide a data set documenting state machine issues. In particular, we claim to provide state machines showing hosts ignoring messages, invalid message paths leading to a completed TLS handshake, and padding-dependent behavior possibly enabling a padding oracle attack. These examples correspond to findings described in Sections 7.1.1 to 7.1.3 of our main paper.

A.4.2 Experiments

(E1): *Full execution for OpenSSL 3.4.0 [10 human-minutes + 5 compute-hours]: this experiment demonstrates state learning and analysis for a modern, correctly-behaving TLS implementation.*

Preparation: Ensure Docker images are built via `setup.sh`. If the 5-hour computational step was not run during setup, `E1.sh` will execute it.

Execution: Run `./experiments/E1.sh` from the repository directory. The script will:

- Check if the state machine already exists from setup
- If not, run the 5-hour learning process
- Execute the automated analysis
- Demonstrate tracing through the state machine with duplicate ClientHellos and unexpected certificates

Results: The automated analysis should report no issues beyond Internal Error alerts. The script demonstrates:

- Correct rejection of duplicate ClientHello with Unexpected Message alert
- Proper rejection of unsolicited client certificates
- A simplified visualization of the obtained state machine should further be available at `experiment_outputs/E1/alphabet-13/OpenSSL3.4.0_short.pdf`
- A simplified visualization for a smaller alphabet, resulting in a less extensive state machine, should further be available at `experiment_outputs/E1/alphabet-1/OpenSSL3.4.0_short.pdf`
- A log file providing stats for the extraction of the individual alphabets should be available at `experiment_outputs/E1/app.log`. For the individual alphabets, a high cache efficiency of >97% should be indicated.

(E2): *Illustrating issue detection based on OpenSSL 1.0.1j [10 human-minutes + 10 compute-minutes]: this experiment demonstrates the automated detection of state machine vulnerabilities in an older OpenSSL version. The issue is similar to the NetScaler real-world finding presented in the paper. Note that the execution is scoped to one alphabet to reduce the execution time.*

Preparation: Ensure Docker images are built via `setup.sh`. If the 10-minute computational step was not run during setup, `E2.sh` will execute it.
Execution: Run `./experiments/E2.sh` from the repository directory. The script will:

- Check if the state machine already exists from setup
- If not, run the 10-minute learning process (limited to the first alphabet as this suffices to illustrate the issue)
- Execute the automated analysis
- Demonstrate the issue by tracing duplicate ClientHellos
- Show complete invalid handshake paths involving duplicate ClientHellos

Results: The analysis identifies multiple issues:

- Invalid message paths with duplicate ClientHellos that complete the handshake
- Illegal inputs (like CCS after handshake) that do not trigger an error
- The script demonstrates how duplicate ClientHellos incorrectly receive ServerHello responses twice
- A simplified visualization of the obtained state machine should further be available at `experiment_outputs/E2/alphabet-1/OpenSSL1.0.1j_short.pdf`

(E3): *Inspecting key findings from our dataset [15 human-minutes]: this experiment analyzes representative state machines from our dataset to demonstrate selected issues we observed.*
Preparation: Ensure Docker images are built and dataset is extracted via `setup.sh`.
Execution: Run `./experiments/E3.sh` from the repository directory. The script analyzes three state machines:

- `completed-59.xml` - A NetScaler state machine which accepts duplicated ClientHello messages
- `completed-331.xml` - A state machine accepting unsolicited certificates
- `completed-1280.xml` - A state machine exhibiting deviating behavior for multiple padding oracle test vectors

Results: The script demonstrates:

- NetScaler accepting duplicate ClientHellos (similar to E2)
- Improper certificate handling allowing unrequested certificates to be sent in the handshake
- Padding oracle vulnerabilities with behavioral differences for different padding types
- Each analysis includes traces showing the specific message paths leading to issues

A.5 Notes on Reusability

Scope In both E1 and E2, we configure the Docker container to run our state learner tool to perform 20,000 random word queries per state when conducting the equivalence tests. In our study, we used 42,000 random queries. To conduct the experiment with the same extent of equivalence tests, the `-queries` parameter of the respective shellscript can be adjusted. For E2, we further limit the execution to the first alphabet. Deleting the `-alphabetLimit` parameter from the shellscript will result in a full execution. Note that extracting the full state machine of OpenSSL 1.0.1j takes significantly longer than extracting the state machine of OpenSSL 3.4.0.

Inspecting the Dataset To inspect more of the state machines we collected, please refer to any of the Docker run commands for the CLI tool given in `experiments/E3.sh` and adapt the file path (`-f`) to point to another state machine XML file.

Tools For applying our state learner to other targets, we recommend to use the flags from E1 or E2 as guidelines as these parameters reflect how we used the tool for our study. Additionally, both the state learner and the state machine analysis tool provide a brief help functionality to guide users through their features.

- **State Learner**: Access help by running the tool with the `-h` or `-help` flag. This displays all available command-line options, including configuration parameters for alphabet selection, learning algorithms, and output formats.

- **State Machine Analysis Tool**: Once in the interactive shell, type `help` to see all available commands. For detailed information about a specific command, use `help <command>`. Additionally, launching the tool with `-help` provides command-line usage options.

Building Without Docker Building the project outside of the Docker image should only require Maven and a Java Development Kit. Please note that Java 11 is required to run the resulting Jars as some dependencies used in our project are not compatible with newer Java versions.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at `https://secartifacts.github.io/usenixsec2025/`.