TLS-Anvil: Adapting Combinatorial Testing for TLS Libraries

Marcel Maehren and Philipp Nieting, *Ruhr University Bochum;*
Sven Hebrok, *Paderborn University;* Robert Merget, *Ruhr University Bochum;*
Juraj Somorovsky, *Paderborn University;* Jörg Schwenk, *Ruhr University Bochum*

https://www.usenix.org/conference/usenixsecurity22/presentation/maehren

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 31st USENIX Security Symposium.

August 10–12, 2022 • Boston, MA, USA

978-1-939133-31-1

Open access to the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium is sponsored by USENIX.

A Artifact Appendix

A.1 Abstract

TLS-Anvil is a test suite that evaluates the RFC compliance of Transport Layer Security (TLS) libraries using combinatorial testing (CT). To facilitate the automated analysis of multiple TLS libraries, we additionally provide the TLS-Docker-Lib project, which contains around 700 images for various versions of 22 TLS libraries. The test results composed by TLS-Anvil for the libraries BearSSL, BoringSSL, Botan, GnuTLS, LibreSSL, MatrixSSL, mbed TLS, NSS, OpenSSL, Rustls, s2n, tlslite-ng, and wolfSSL are the foundation of the evaluation in our paper. The results can be reproduced by running TLS-Anvil against local installations or docker images of the libraries. TLS-Anvil and its dependencies are written in Java, specifically for Java 11. Hardware requirements depend on the desired extent of the combinatorial testing (test strength). 16 GB of RAM are sufficient to test each library with strength three as we did for the paper.

A.2 Artifact check-list (meta-information)

- **Compilation:** TLS-Anvil and its dependencies can be built using Maven with Java 11.
- **Binary:** We provide executable jars built for Java 11 for TLS-Anvil here . TLS-Anvil is also provided as Docker image that is distributed using GitHub Packages. The TLS server/clients that we evaluated with TLS-Anvil are part of the TLS-Docker-Library. Those are designed as Docker images as well, but not available yet. However, the images can be built locally.
- **Data set:** We provide our raw test results to be used with the Report Analyzer here
- **Run-time environment:** We tested our artifacts using Linux or macOS. Since the TLS-Docker-Library contains some bash and python scripts this is also our recommendation. Those scripts depend on Docker. Therefore, root access is also needed.
- **Hardware:** No special Hardware needed. 16GB Ram is recommended to run TLS-Anvil.
- **Output:** The results of TLS-Anvil are stored in various json files. Those should be processed and evaluated with our Report Analyzer web application, which is also available as Docker image.
- **Experiments:** Analysis of TLS clients and servers using TLS-Anvil.
- **Required disk space:** Roughly 50 GB, considerably more if the whole Docker library is built (roughly 1 TB).
- **Approximate time required to prepare the workflow:** 2h
- **Time required to complete experiments:** Key results can be recreated quickly using a low testing strength of one, which takes around one hour for OpenSSL. Testing with strength two takes around six hours and strength three around 31 hours. While we evaluated with a strength of three, we identified that all findings can already be found using a strength of two. Generating all results for all libraries using docker images takes around one week when evaluating two libraries in parallel. Table 4 in our paper contains an overview of execution times.
- **Publicly available evolving repo:** The TLS-Anvil GitHub repo is available here.
- **Code licenses:** Apache 2
- **Data licenses:** Apache 2
- **Archived stable references:** The archived tags for TLS-Anvil, TLS-Docker-Library, and the Large-Scale-Evaluator are:
  - TLS-Anvil:
    Tag v1.0.3
  - TLS-Docker-Library:
    Tag 2.0.1
  - TLS-Anvil-Large-Scale-Evaluator:
    Tag 1.0.1

  The required versions of TLS-Anvil's dependencies are listed as submodules of the git repository.

A.3 Description

A.3.1 How to access

- TLS-Anvil can be found here.
- TLS-Docker-Library can be found here.
- TLS-Large-Scale-Evaluator can be found here.

Dependencies (optional)

- TLS-Attacker can be found here.
- TLS-Scanner can be found here.

See submodules of tags given in 'Archived' from subsection A.2 for specific versions.

A.3.2 Hardware dependencies

Depending on the desired testing strength, up to 16 GB of RAM are required. The overall CPU load of the system may affect the tests performed by TLS-Anvil. Please ensure that the system can provide enough processing resources for TLS-Anvil to obtain accurate results. A relatively recent CPU should be enough.

A.3.3 Software dependencies

To evaluate the considered libraries with TLS-Anvil and TLS-Docker-Lib, Java 11, Docker, Docker-Compose, and Maven are required. To build TLS-Anvil and its dependencies without Docker, Java 11 SDK is required. Running TLS-Anvil outside of a Docker container requires tcpdump.

A.3.4 Data sets

We provide the raw outputs of TLS-Anvil for the libraries considered in our evaluation. While we evaluated the libraries using a test strength of up to three, we identified that all findings can already be reproduced with a test strength of two. We hence provide the outputs for testing strength two.

A test output consists of multiple json files. These should be processed using our web application that visualizes the results (see below for more details).

A.3.5 Models

N/A

A.3.6 Security, privacy, and ethical concerns

N/A

A.4 Installation

Setting up TLS-Anvil You can either use our provided Docker images or build everything yourself using a Dockerfiles contained in the TLS-Anvil repository. Note that the docker commands below fetch the latest version of TLS-Anvil.

a) Using the provided docker images:

1. Pull TLS-Anvil Docker image

```
docker pull \
ghcr.io/tls-attacker/tlsanvil:latest
```

2. Pull Report Analyzer Docker image

```
docker pull \
ghcr.io/tls-attacker/\
tlsanvil-reportanalyzer:latest
```

3. Pull Report Uploader Docker image

```
docker pull \
ghcr.io/tls-attacker/\
tlsanvil-result-uploader:latest
```

4. Adjust the tags to match a local build

```
docker tag \
ghcr.io/tls-attacker/tlsanvil:latest \
tlsanvil:latest
docker tag \
ghcr.io/tls-attacker/\
tlsanvil-reportanalyzer:latest \
uploader:latest
docker tag \
ghcr.io/tls-attacker/\
tlsanvil-reportanalyzer:latest \
reportanalyzer:latest
```

5. Clone the TLS-Anvil repository

```
git clone \
https://github.com/tls-attacker/\
TLS-Anvil.git
```

b) Building TLS-Anvil and the Report Analyzer yourself:

1. Clone the TLS-Anvil repository

```
git clone \
https://github.com/tls-attacker/\
TLS-Anvil.git
```

2. Run the build script

```
cd TLS-Anvil/
sh build.sh
```

3. Build the Report-Analyzer Docker image

```
cd Report-Analyzer/
docker-compose build
```

4. Build the upload Docker image (this uploads TLS-Anvil json files to the Report Analyzer web application)

```
cd src/backend/uploader
docker build -t uploader .
```

Setting up TLS-Docker-Library

1. Clone the repository using

```
git clone https://github.com/tls-attacker/\
TLS-Docker-Library.git
```

2. Navigate to TLS-Docker-Library/

3. Execute `setup.sh`

4. Execute `mvn install -DskipTests`

Downloading OpenSSL docker images

1. Get provided client and server images

```
docker pull ghcr.io/tls-attacker/\
openssl-client:1.1.1i
docker pull ghcr.io/tls-attacker/\
openssl-server:1.1.1i
```

2. Adjust the tags to match a local build

```
docker tag 4791200dbed9 \
openssl-server:1.1.1i
docker tag 8fe8f5106aa9 \
openssl-client:1.1.1i
```

Building an OpenSSL library Docker Container yourself (optional)

1. Navigate to TLS-Docker-Library/images/
2. Build the OpenSSL 1.1.1i server and client image

```
python3 build-everything.py -l \
openssl -v 1.1.1i
```

Building the TLS-Anvil-Large-Scale-Evaluator (optional)

1. Clone the repository

```
git clone https://github.com/tls-attacker/\
TLS-Anvil-Large-Scale-Evaluator.git
```

2. Navigate to TLS-Anvil-Large-Scale-Evaluator/
3. Run `mvn install -DskipTests`

A.5 Experiment workflow

TLS-Anvil can be run in client and server test mode depending on the tested endpoint. Regardless of the endpoint, TLS-Anvil first performs a feature discovery to determine suitable values for test parameters as well as applicable test templates. Since most test templates are either exclusively client or server test templates, many tests will be skipped during the execution. This is also the case for tests that must be skipped if an endpoint does not support a feature required for the test.

During the execution, TLS-Anvil creates json files for every executed test template containing the detailed results. The following guide shows how the OpenSSL server and client can be tested using TLS-Anvil. Both peers run inside a Docker container. The test results are stored inside the current working directory.

A.5.1 Testing OpenSSL Server

1. Create a separate Docker network

```
docker network create tls-anvil
```

2. Start the OpenSSL Server

```
docker run \
-d \
--rm \
--name openssl-server \
--network tls-anvil \
-v cert-data:/certs/ \
openssl-server:1.1.1i \
-port 8443 \
-cert /certs/rsa2048cert.pem \
-key /certs/rsa2048key.pem
```

3. Start TLS-Anvil

```
docker run \
--rm \
-it \
-v $(pwd):/output/ \
--name tls-anvil \
--network tls-anvil \
tlsanvil:latest \
-outputFolder ./ \
-parallelHandshakes 1 \
-strength 1 \
-identifier openssl-server \
server \
-connect openssl-server:8443 \
-doNotSendSNIExtension
```

A.5.2 Testing OpenSSL Client

1. Create a separate Docker network

```
docker network create tls-anvil
```

2. Start TLS-Anvil

```
docker run \
--rm \
-it \
-v $(pwd):/output/ \
--network tls-anvil \
--name tls-anvil \
tlsanvil:latest \
-outputFolder ./ \
-parallelHandshakes 3 \
-parallelTests 3 \
-strength 1 \
-identifier openssl-client \
client \
-port 8443 \
-triggerScript curl --connect-timeout 2
 ↪  openssl-client:8090/trigger
```

3. Start OpenSSL Client

```
docker run \
-d \
```

```
--rm \
--name openssl-client \
--network tls-anvil \
openssl-client:1.1.1i \
-connect tls-anvil:8443
```

A.5.3 Testing all Libraries

Since we needed to analyze multiple servers and clients of different libraries the manual process shown above results in a large overhead. Therefore, we automated the Docker container launching with a (Java) tool TLS-Anvil-Large-Scale-Evaluator. To analyze the OpenSSL server using this tool, the following command should be executed from the main folder of the cloned repository:

```
java -jar \
apps/TLS-Anvil-Large-Scale-Evaluator.jar \
-m server -e testsuite -i openssl -v 1.1.1i \
-p 1 -s 2
```

To analyze the client use:

```
java -jar \
apps/TLS-Anvil-Large-Scale-Evaluator.jar \
-m client -e testsuite -i openssl -v 1.1.1i \
-p 1 -s 2
```

After building the docker images for the libraries versions listed in the paper, you can run TLS-Anvil against all considered implementations using:

```
-i bearssl boringssl botan gnutls libressl \
mbedtls nss openssl rustls s2n tlslite_ng \
wolfssl matrixssl \
-v 0.6 3945 2.17.3 3.7.0 3.2.3 2.25.0 3.60 \
1.1.1i 0.19.0 0.10.24 0.8.0-alpha39 \
4.5.0-stable 4.3.0
```

A.6 Evaluation and Expected Results

To evaluate our results, you can either recreate them using the explanations from subsection A.5 or use the data from our experiments, which are available here.

A.6.1 Uploading Test Results to the Report Analyzer

To start with the evaluation you should start the Report Analyzer by navigating into the cloned TLS-Anvil repository and running:.

```
cd Report-Analyzer
docker-compose up -d
```

After that, a web application should be available at `http://localhost:5000`. The application offers three main pages: 'Upload', 'Analyzer', and 'Manage'. While it is possible to upload results using the web app, we recommend using the uploader Docker container as it collects all necessary files automatically. To start a recursive search for results from your current directory and upload them to the web application, use:

```
docker run \
--rm \
-it \
--network host \
-v $(pwd):/upload \
uploader
```

A.6.2 Analyzing Test Results

The drop down menu in the upper left corner of the 'Analyzer' page shows all test results uploaded to the Report Analyzer's database. In order to analyze a result, select a test result and click the 'Add' button next to the drop down menu. The Report Analyze will then show some basic execution properties, such as the execution time, and a list containing each test template that was executed and a symbol that indicates the test result. A check mark indicates a strictly succeeded test, a cross indicates a failed test. A check mark with a warning sign indicates a *conceptually* succeeded test, a cross with a warning sign indicates a *partially* failed test. An exclamation mark indicates further information is available on the test results.

Clicking on the result symbol leads to a detail page for the test template. The list shows the result for each test case executed throughout the test template. Clicking on an identifier on the left or a sub result on the right opens a modal window that summarizes the test inputs used for this specific test case as well as some meta data. Additionally, it is possible to inspect and download a pcap file that contains the recorded traffic of this specific test case.

Using the filter menu above the results table, it is possible to filter connections, for example, to only show connections where a parameter had a specific value or where a specific test result has been determined.

A.6.3 Key Results

Using TLS-Anvil, we identified a variety of RFC violations for the 13 considered libraries. Below, we describe how to identify some of the main findings using the Report Analyzer and our provided test results.

wolfSSL Authentication Bypass wolfSSL client 4.5.0 allows a server to bypass the authentication by sending a *Certificate* message with an empty certificate list. Open the test results for wolfSSL client (wolfssl-client-4.5.0-stable-WWkfM) in the Report Analyzer. Search for the

`emptyCertificateList` test in the list and click on the result symbol on the right. On the new page, some connections are marked as succeeded (with a checkmark), while others failed. The seemingly succeeded tests are the result of wolfSSLs intolerance for some record lengths. Since record fragmentation with different lengths is a parameter of our test input, wolfSSL sometimes can not finish the handshake. Since our RFC violation and the first fragmented records both take place within our first flight of messages, it appears as if wolfSSL sometimes correctly rejects our malformed *Certificate* message which is not the case. To obtain a clear test result, click on the drop down menu on the top of the page and select `RECORD_LENGTH`. This will filter the connections to only show a specific fragmentation length in bytes. Change the value on the right from 1 to 16384, which effectively shows only those handshakes where no record fragmentation was used. Inspecting the remaining test results of the list, either by hovering on the result symbol or clicking on it, shows that wolfSSL always failed to reject the invalid message and instead proceeded to send its final handshake message and application data. You can compare this behavior to the very similar `emptyCertificateMessage` test, where wolfSSL behaves as expected. In contrast to the previous test, here we use an entirely empty message (with a message length of zero). By applying the same filter steps as before, the results show that wolfSSL rejects `this type` of empty *Certificate* message correctly.

MatrixSSL Padding Oracle Vulnerability The MatrixSSL 4.3.0 client indicates an invalid padding upon decryption for ciphersuites that use SHA256 to compute the HMAC. Open the same test result as before and search for the test `invalidCBCPadding`. MatrixSSL aborted the connection for all messages that contained an invalid padding value. However, for SHA256 cipher suites, MatrixSSL does not send an alert as it does for all other cipher suites. We further analyzed this behavior and identified that this is due to a segmentation fault. This behavior is unique to the invalid padding error case and thus leaks information about the obtained plaintext. You can compare this behavior to the `invalidMAC` test, where MatrixSSL always sends an alert regardless of the cipher suite.

MatrixSSL Unproposed Groups The MatrixSSL 4.3.0 client accepts that a server negotiates certain curves that have not been proposed by the client. While MatrixSSL offers the curves secp256r1, secp384r1, x25519, and secp521r1 it also accepts *ServerKeyExchange* messages that contain a public key of the curves secp192r1 and secp224r1, which both have significantly weaker security properties. To identify this behavior, open the test results for MatrixSSL client (matrixssl-client-4.3.0-ik8fF) and search for the test `acceptsUnproposedNamedGroup` and click on the test result symbol. Using the drop down menu at the top, select 'Test Result' as the filter and set the desired value to 'FAILED' in the drop down menu on the right. By clicking on the remaining test results, the Report Analyzer shows a text box with a summary of details in json. First of all, the `DerivationContainer` element shows the chosen parameters of the test. The `NAMED GROUP` parameter for the failing tests is either secp192r1 or secp224r1. Further below, the `Stacktrace` shows the failed JUnit Assertion with an indication of the error. In this case, an alert was expected (since the server made an illegal selection) but MatrixSSL client proceeded to send a *ClientKeyExchange*, *ChangeCipherSpec*, and *Finished* message instead.

A.7 Experiment customization

You can also run your own experiments with TLS-Anvil against any server or client. For this purpose run the jar available in TLS-Anvil/TLS-Testsuite/apps from the cloned and built repository or use our provided jars. To test a server running on localhost:4433, use:

```
java -jar TLS-Testsuite.jar server -connect \
localhost:4433
```

To test a client from port 4433, use:

```
java -jar TLS-Testsuite.jar client -port 4433 \
-triggerScript triggerScript.sh
```

Where `triggerScript.sh` contains the command to start a client that connects to localhost:4433.

A.8 Notes

Analyzing the issues for a scientific paper sometimes required additional manual labour, as we grouped failed tests based on their route cause to get to the final number of findings. Therefore the number of failed tests is larger than the number of findings.

A.9 Version

Based on the LaTeX template for Artifact Evaluation V20220119.