Exploring the Unknown DTLS Universe: Analysis of the DTLS Server Ecosystem on the Internet

Nurullah Erinola and Marcel Maehren, *Ruhr University Bochum;* Robert Merget, *Technology Innovation Institute;* Juraj Somorovsky, *Paderborn University;* Jörg Schwenk, *Ruhr University Bochum*

https://www.usenix.org/conference/usenixsecurity23/presentation/erinola

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Open access to the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium is sponsored by USENIX.

Exploring the Unknown DTLS Universe: Analysis of the DTLS Server Ecosystem on the Internet

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

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

A Artifact Appendix

A.1 Abstract

TLS-Scanner is an open-source tool to assist pentesters and security researchers in evaluating TLS server implementations. It automatically scans a TLS server and provides a report of supported features like protocol versions, cipher suites, extensions, and potential security issues.

In our work, we extended TLS-Scanner with support for DTLS and implemented additional tests specifically designed to evaluate DTLS-specific features. Subsequently, we evaluated twelve open-source DTLS server implementations and uncovered eleven security vulnerabilities. We then proceeded to scan publicly available DTLS servers to gain detailed insights into the publicly accessible DTLS server landscape.

Artifact users can reproduce the results of our lab evaluation by running TLS-Scanner against the respective DTLS server implementations.

A.2 Description & Requirements

Upon completion of the scan, TLS-Scanner provides a comprehensive report containing detailed information regarding the server's configuration and its security-relevant properties. We provide the source code of the extended TLS-Scanner and Docker files of OpenSSL and Mbed TLS as artifacts, enabling the testing of TLS-Scanner.

A.2.1 Security, Privacy, and Ethical Concerns

We are not aware of any exploitable issues in TLS-Scanner. TLS-Scanner only establishes multiple DTLS connections to the server under test. However, depending on the number of threads, it is possible to overwhelm the server. Therefore, by default, the scan is performed with one thread. Note that such a scan will inevitably reveal your IP address to the tested server.

TLS-Scanner can also be used to scan servers owned by other people. Depending on your local jurisdiction, it may be illegal for you to do so. Additionally, conducting scans on public servers should follow the best practices for Internet-wide scanning setup by Durumeric et al. [1].

A.2.2 How To Access

Our artifact can be found on GitHub at https://github.com/tls-attacker/Exploring-the-Unknown-DTLS-Universe/tree/563b9ca12920eed26b00f518fe7465b2b833024e. The repository includes the source code of the extended TLS-Scanner and example Docker files which build open-source DTLS server implementations (OpenSSL and Mbed TLS).

A.2.3 Hardware Dependencies

None.

A.2.4 Software Dependencies

TLS-Scanner is written in Java. This requires Maven and Java 11 to be installed. To run the example servers in the docker containers, Docker is required. We tested this artifact on Kubuntu 22.04, but any Linux system should work. TLS-Scanner should also run on Windows (but the docker examples will not).

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

Downloading the Artifact. Clone the GitHub repository using *git*:

```
https://github.com/tls-attacker/\
Exploring-the-Unknown-DTLS-Universe/tree/\
563b9ca12920eed26b00f518fe7465b2b833024e
```

Installing Java and Maven. Install Java 11 and Maven using *apt*:

```
sudo apt install openjdk-11-jdk
sudo apt install maven
```

To verify if the dependencies are installed and set up correctly, run the following commands:

```
java -version
mvn -version
```

If everything works correctly, both commands should display their respective versions. Additionally, it is important to ensure that the correct version of Java is specified for Maven.

Installing Docker. Install Docker by following the instructions at https://docs.docker.com/engine/install/.

Setting up TLS-Scanner.

a) Using the provided Dockerfile:

1. Navigate to `tls-scanner/`
2. Run
   ```
   docker build --tag tls-scanner --file \
   dockerfile-tls-scanner .
   ```
3. Run `docker images`

   If everything works correctly, the last command should print the names of all available docker images on your system. The output should contain *tls-scanner*.

b) Building TLS-Scanner yourself:

1. Navigate to `tls-scanner/TLS-Attacker/`
2. Execute `mvn clean install`
3. Navigate to `tls-scanner/TLS-Scanner/`
4. Execute `mvn clean package`

   If everything compiles correctly, the `apps\` folder should now contain the `TLS-Server-Scanner.jar` file.

Building the Server Implementations.

1. Navigate to `libraries/`
2. Execute `setup.sh`
3. Run `docker images`

If everything works correctly, the last command should again print the names of all available docker images on your system. The output should contain the names of the server implementations (e.g., *openssl-dtls-server* or *mbedtls-dtls-server*).

A.3.2 Basic Test

Testing TLS-Scanner. To verify that the TLS-Scanner can be executed correctly, run the following command:

```
docker run --rm --network="host" --name \
tls-scanner tls-scanner -help
```

If everything works correctly this should print the parameter list of TLS-Scanner with usage instructions.

Testing the Server Implementations. To verify that the docker images can be used, run the following command:

```
docker run --rm --network="host" --name \
mbedtls-dtls-server mbedtls-dtls-server \
server_port=4433 dtls=1
```

If everything works correctly this should start the example server of Mbed TLS.

A.4 Evaluation workflow

Running TLS-Scanner on a DTLS server implementation is straightforward. Typically, the DTLS server is started, then TLS-Scanner is started. TLS-Scanner will then perform the scan without interaction from the user. After completing the scan, TLS-Scanner will output the results which the user can analyze.

A.4.1 Major Claims

We claim to be able to evaluate various DTLS-specific features of a given server, including potential DoS vulnerabilities. Specifically, we claim to evaluate the properties of twelve open-source DTLS server implementations. In the following, we exemplary describe our claims for OpenSSL and Mbed TLS:

(C1): OpenSSL issues 20 byte long cookies and deviates from the recommended cookie computation.
(C2): OpenSSL does not perform a cookie exchange upon renegotiation.
(C3): OpenSSL allows users to implement *no* cookie exchange, a *stateful* cookie exchange, or a *stateless* cookie exchange. The example server uses the stateful cookie exchange by default, but stateless mode can be requested through command line parameters. To mitigate DoS attacks, the stateless mode should be used when deploying OpenSSL in production.

(C4): Mbed TLS issues 32 byte long cookies and deviates from the recommended cookie computation.
(C5): Mbed TLS supports the fragmentation of messages after the cookie exchange is successfully completed.

The experiment (E1) will demonstrate (C1)-(C3) while (C4)-(C5) are demonstrated by the experiment (E2). The results of the two experiments are summarized in our paper in Table 1 with a detailed explanation in Section 5.

A.4.2 Experiments

Since TLS-Scanner is designed to scan only a single server, we define one scan per experiment.

(E1 - OpenSSL) *(5 human-minutes + 10 compute-minutes)* In this experiment, the OpenSSL example server is scanned with TLS-Scanner. TLS-Scanner will then output a report.

1. Start the OpenSSL server

```
docker run --rm -v [absolute path to \
libraries/certs/]:/certs/ \
--network="host" --name \
openssl-dtls-server openssl-dtls-server \
-key /certs/private_key.pem -cert \
/certs/certificate.pem -accept 4433 -dtls
```

2. Start the evaluation with TLS-Scanner

```
docker run --rm --network="host" --name \
tls-scanner tls-scanner -connect \
localhost:4433 -dtls -timeout 100
```

TLS-Scanner will now perform a series of DTLS handshakes. After that, the report should be visible.

Results.
(C1): The report should contain a section `DTLS Hello Verify Request`. It summarizes the behavior the server showed against the implemented cookie exchange tests. There you can see which client parameters influence the cookie computation. For OpenSSL, it should contain:

```
DTLS Hello Verify Request

HVR Retransmissions          : false
Cookie length                : 20
Checks cookie                : true
Cookie is influenced by
-ip                          : not tested yet
-port                        : true
-version                     : false
-random                      : false
-session id                  : false
-cipher suites               : false
-compressions                : false
```

To confirm (C1), the `Cookie length` field should have the value `20`. In addition, the `-version`, `-random`, `-session id`, `-cipher suites`, and `-compressions` fields should have `false`. Please note that evaluating if the IP influences the cookie computation requires a proxy running on a host with a different IP.

(C2): The report should contain a section `Renegotioation`. It summarizes whether the server supports renegotiation and whether cookie exchange is performed there. For OpenSSL, it should contain:

```
Renegotioation

Secure (Extension)           : true
Secure (CipherSuite)         : true
Insecure                     : false
DTLS cookie exchange in renegotiation : false
```

To confirm (C2), the `DTLS cookie exchange in renegotiation` field should have `false`.

(C3): The report should contain a section `DTLS Fragmentation`. It summarizes the behavior the server showed against the implemented fragmentation tests. For OpenSSL, it should contain:

```
DTLS Fragmentation

Supports fragmentation       : true
Supports fragmentation with individual transport packets :
    ↪ true
```

To confirm (C3), both fields should have `true`.

(E2 - Mbed TLS) *(5 human-minutes + 10 compute-minutes)* In this experiment, the Mbed TLS example server is scanned with TLS-Scanner. TLS-Scanner will then output a report.

1. Start the Mbed TLS server

```
docker run --rm --network="host" --name \
mbedtls-dtls-server mbedtls-dtls-server \
server_port=4433 dtls=1
```

2. Start the evaluation with TLS-Scanner

```
docker run --rm --network="host" --name \
tls-scanner tls-scanner -connect \
localhost:4433 -dtls -timeout 100
```

TLS-Scanner will now perform a series of DTLS handshakes. After that, the report should be visible.

Results.
(C4): Similar to (C1). For Mbed TLS, the `DTLS Hello Verify Request` section should contain:

```
DTLS Hello Verify Request

HVR Retransmissions          : false
Cookie length                : 32
Checks cookie                : true
Cookie is influenced by
-ip                          : not tested yet
```

```
-port                          : false
-version                       : cannot be tested
-random                        : false
-session id                    : false
-cipher suites                 : false
-compressions                  : false
```

To confirm (C4), the `Cookie length` field should have the value `32`. In addition, the `-port`, `-random`, `-session id`, `-cipher suites`, and `-compressions` fields should have `false`. Please note that evaluating if the version influences the cookie cannot be executed for Mbed TLS because it supports only one protocol version (DTLS 1.2).

(C5): Similar to (C3). For Mbed TLS, the `DTLS Fragmentation` section should contain:

```
DTLS Fragmentation

Supports fragmentation       : partially
-After cookie exchange
Supports fragmentation with individual transport packets :
    ↪ partially
-After cookie exchange
```

To confirm (C5), both fields should have `partially`.

A.5 Notes on Reusability

Our extensions to TLS-Scanner have been merged in the project and released with v5.2.5. It can be found on GitHub at https://github.com/tls-attacker/TLS-Scanner/releases/tag/v5.2.5. In addition, our extensions to TLS-Attacker which is used by TLS-Scanner have been merged in the project and are contained in the latest release v5.2.1. It can be found on GitHub at https://github.com/tls-attacker/TLS-Attacker/releases/tag/v5.2.1.

To perform large-scale scans with TLS-Scanner, TLS-Crawler can be used. It utilizes multiple TLS-Scanner instances to scan a large number of servers in parallel and write the results to a database. The latest release v1.0.1 can be found on GitHub at https://github.com/tls-attacker/TLS-Crawler/releases/tag/v1.0.1.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20220926. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2023/.

References

[1] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. ZMap: Fast Internet-wide scanning and its security applications. In *22nd USENIX Security Symposium*, 2013.