USENIX Security '25 Artifact Appendix: Towards Practical, End-to-End Formally Verified X.509 Certificate Validators with Verdict

Zhengyao Lin[†] Michael McLoughlin[†] Pratap Singh[†] Rory Brennan-Jones[‡]
Paul Hitchcox[†] Joshua Gancher[§] Bryan Parno[†]
[†]*Carnegie Mellon University* [‡]*University of Rochester* [§]*Northeastern University*

A Artifact Appendix

A.1 Abstract

Paper: Validating X.509 certificates is a critical part of Internet security, but the relevant standards are complex and ambiguous. Most X.509 validators also intentionally deviate from these standards in idiosyncratic ways, often for security or backward compatibility. Unsurprisingly, the result is a long history of security vulnerabilities.

In this work, we present Verdict, the first end-to-end formally verified X.509 certificate validator with customizable validation policies. Verdict's formal guarantees cover certificate parsing, path building, and path validation. To make Verdict practical to both verify and to use, we specify its correctness generically in terms of a user-supplied validation policy written concisely in first-order logic, with a proof-producing compiler to efficient Rust code.

To demonstrate Verdict's expressiveness, we use Verdict's policy framework to implement the X.509 validation policies in Google Chrome, Mozilla Firefox, and OpenSSL, and formally prove that they conform to a subset of RFC requirements. We instantiate Verdict with each policy and show that Verdict matches the corresponding baseline's behavior and state-of-the-art performance on over ten million certificates from Certificate Transparency logs.

Artifact: This artifact includes the source code of Verdict, as well as forks of six other X.509 validators we evaluate against: Chrome, Firefox, OpenSSL, ARMOR, CERES, and Hammurabi. We also provide scripts to automate all three main evaluations we perform in the paper.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Verdict is a verified X.509 certificate validator, and as such, it does not pose any immediate security, privacy, or ethical concerns. We also provide a self-contained and compiled version of the artifact as a Docker image, which adds another layer of isolation to avoid changes to your host system.

A.2.2 How to access

The artifact can be accessed via the following URLs:

- Artifact image: https://doi.org/10.5281/zenodo.15468400
- Main Verdict repository: https://github.com/secure-foundations/verdict
- Evaluation tools and build scripts of the artifact image: https://github.com/secure-foundations/verdict-bench

The first Docker image is sufficient for artifact evaluation.

A.2.3 Hardware dependencies

Our artifact image requires an x86-64 machine with at least 16 GiB of RAM and 20 GB of free disk space.

A.2.4 Software dependencies

We recommend using Ubuntu (at least 20.04) with Docker 27.5.1. Newer versions may work too. Your host user must also be able to use the `--cap-add=NET_ADMIN` flag in Docker, in order to perform the end-to-end evaluation with Rustls.

A.2.5 Benchmarks

In the paper, we use about 10M certificate chains from Certificate Transparency (CT) logs. For convenience of evaluation, we only include a sampled subset of 35,000 chains in the artifact image. You are also welcome (but not required) to use your own set of certificate chains, and the instructions to do so are in the `README.md` file in the evaluation tools repository.

All other datasets are included in the artifact image.

A.3 Set-up

From this point onwards, we assume that you are using the pre-build artifact image (Section A.2.2).

A.3.1 Installation

Download the Docker image `verdict-bench-image.tar.gz` (Section A.2.2). Then load the image by

```
docker load --input verdict-bench-image.tar.gz
```

Alternatively, you can also pull the image directly from GitHub for faster access:

```
docker pull \
    ghcr.io/secure-foundations/verdict-bench && \
docker tag \
    ghcr.io/secure-foundations/verdict-bench \
    verdict-bench
```

A.3.2 Basic Test

Start the container by:

```
docker run -it --cap-add=NET_ADMIN verdict-bench
```

Then run `make test` to perform all benchmarks but without multiple samples to reduce noise.

Finally, `make figures` should display a list of LaTeX tables corresponding to Figures 6, 7, and 8 in the paper. The plot for Figure 5 can also be found at `results/performance.pdf`. Note that the performance results may be very noisy and inaccurate at this point, and you should restart the container and run `make eval` to obtain more accurate results (see Section A.4.2).

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): *Verdict's performance is on par with state-of-the-art X.509 validators (Chrome, Firefox, OpenSSL) on a large set of real-world certificates, and it is orders of magnitude faster than prior academic work on more trustworthy X.509 validation (ARMOR, CERES, Hammurabi).*

(C2): *Verdict has high fidelity formal models of Chrome, Firefox, and OpenSSL when compared on the x509-limbo test suite.*

(C3): *When integrated into Rustls, Verdict causes negligible performance overhead in end-to-end HTTPS requests on popular websites.*

(C4): *Verdict is a formally verified X.509 certificate validator with an end-to-end specification.*

A.4.2 Experiments

The following experiments are used to justify the major claims above. In particular, **(E1)** performs benchmarks and tests to demonstrate **(C1)**, **(C2)**, and **(C3)**; **(E2)** builds/verifies Verdict and involves some human inspection to justify **(C4)**.

Experiment **(E1)** does not require Internet connection and all required binaries are included in the Docker image; however, the experiment **(E2)** does need network access to download some dependencies like Rust and Verdict's dependencies.

In all of the following experiments, we assume that the basic set-up steps (Section A.3) have been done, and that you are already inside the artifact container.

(E1) [10 human-minutes + 3.5 compute-hours + 1 GB disk] In this experiment, we perform performance benchmarks and differential tests to demonstrate claims **(C1)**, **(C2)**, and **(C3)**.

How to: Run `make eval`. This might take a few hours.

Results: Run `make figures` to display three LaTeX tables that should have comparable results to Figures 6, 7, and 8 in the main paper in terms of the relative performance of the tools. A PDF file produced at `results/performance.pdf` should also contain similar results to Figure 5 in the paper.

Note that since we are testing on a small subset of CT logs (about 35,000 chains out of the 10 million chains), the differential testing results in the "CT" section will have much fewer certificates; although the "Limbo" section should have similar, and slightly improved results than Figure 7.

Some noise in the performance results (Figures 5, 6, and 8) is also expected (likely within 10% of relative performance between tools), due to the smaller data set as well as differences in the testing environment. However, we expect that the results still support our claim that Verdict is comparable with the performance of Chrome, Firefox, and OpenSSL; and that it is orders of magnitude faster than other academic work.

For more fine-grained results, please see CSV files in the `results` folder:

- `results/perf-<tool>.csv`: performance results of `<tool>` on each tested certificate chain, where the columns denote hash of the leaf certificate, the hostname being validated, validation result, and finally multiple samples of the validation time in microseconds.

- `results/limbo-<tool>.csv`: results of running `<tool>` on the x509-limbo test suite, where the columns denote the test name, the expected result (as indicated by x509-limbo), and the actual result.

- `results/end-to-end-{aws-lc,libcrux}.csv`: performance results of simulating HTTPS requests to popular websites using Rustls with Verdict integrated. The first column is the domain being tested, the second column is the X.509 validator used (`default` for the built-in validator in Rustls, `verdict-chrome/firefox/openssl` are Verdict models of Chrome, Firefox, and OpenSSL). Rest of the columns are samples of end-to-end request time.

(E2) [30 human-minutes + 5 compute-minutes] In this experiment, we build and verify Verdict using Verus, and manually inspect the specifications and proofs of Verdict. This is to support claim **(C4)**.

How to: Run `make build-verdict`. Note that this requires network access. This command will install Verus and various dependencies, and then verify Verdict using Verus.

Results: If everything is successful, Verus has verified all proofs in Verdict, and produced the final binary at `verdict/target/debug/verdict`.

To manually inspect the specifications and proofs in Verdict, you can find the main components in these locations:

- `verdict` is the main verified X.509 validation library. It includes implementations of different policies (`verdict/src/policy`) as well as the policy-independent validation procedure (`verdict/src/validator.rs`). In particular, the main specification in Figure 2 of the paper can be found in `verdict/src/validator.rs`, where `spec_validate_x509_base64` and `Query::valid` should be equivalent to the `spec_valid_x509` spec function in the paper.

- Verdict's formal models of Chrome, Firefox, and OpenSSL X.509 policies can be found in `verdict/src/policy`. For example, `verdict/src/policy/openssl.rs` contains the OpenSSL policy written in Verdict's policy DSL.

- `verdict-parser` contains the verified parsers and serializers of X.509 (`verdict-parser/src/x509`) and various ASN.1 components (`verdict-parser/src/asn1`). The main paper mentions an ASN.1 DSL to automatically generate verified parsers and serializers, and an example of this can be found in `verdict-parser/src/x509/tbs_cert.rs` (i.e., the `asn1!` macro invocation).

A.5 Notes on Reusability

The Docker image came bundled with a built Verdict frontend binary, which you can find at `verdict/target/release/verdict`. You can run `verdict/target/release/verdict --help` to see its usage. Currently, the frontend supports parsing X.509 certificates, validating given certificate chains, and various utilities to run the evaluation in the main paper.

As an example, `verdict/verdict/tests/chains` contains some sample certificate chains. You can verify one of them by running:

```
verdict/target/release/verdict validate \
    -t 1735707600 \
    chrome \
    verdict/verdict/tests/roots.pem \
    verdict/verdict/tests/chains/github.pem
```

This should produce output such as:

```
result: ValidationResult { valid: true, ... }
```

The Verdict library can also be easily used in other Rust projects through Cargo:

```
cargo add verdict \
  --git https://github.com/secure-foundations/verdict.git
```

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/.