USENIX Security '25 Artifact Appendix: Vest: Verified, Secure, High-Performance Parsing and Serialization for Rust

Yi Cai[†*] Pratap Singh[‡] Zhengyao Lin[‡] Jay Bosamiya[¶*]
Joshua Gancher[§*] Milijana Surbatovich[†] Bryan Parno[‡]

[¶]*Microsoft Research* [§]*Northeastern University* [‡]*Carnegie Mellon University*
[†]*University of Maryland, College Park*

A Artifact Appendix

A.1 Abstract

Paper: Many software vulnerabilities lurk in parsers and serializers, due to their need to be both high-performance and conformant with complex binary formats. To categorically eliminate these vulnerabilities, prior efforts have sought to deliver provable guarantees for parsing and serialization. Unfortunately, security, performance, and usability issues with these efforts mean that unverified parsers and serializers remain the status quo.

Hence, we present Vest, the first framework for high-performance, formally verified binary parsers and serializers that combines expressivity and ease of use with state-of-the-art correctness and security guarantees, including—for the first time—resistance to basic digital side-channel attacks. Most developers interact with Vest by defining their binary format in an expressive, RFC-like DSL. Vest then generates and automatically verifies high-performance parser and serializer implementations in Rust. This process relies on an extensible library of verified parser/serializer combinators we have developed, and that expert developers can use directly.

We evaluate Vest via three case studies: the Bitcoin block format, TLS 1.3 handshake messages, and the WebAssembly binary format. We show that Vest has executable performance on-par (or better) than hand-written, unverified parsers and serializers, and has *orders of magnitude* better verification performance relative to comparable prior work.

Artifact: This artifact contains the complete implementation of VestLib and VestDSL, as well as a suite of benchmarks on real-world binary formats including Bitcoin, TLS 1.3, and WebAssembly.

[*]Work done in part while at Carnegie Mellon University.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

This artifact is packaged as a Docker image, which contains the source code of VestLib and VestDSL, the dependencies (e.g., Rust and Verus), and the benchmarks for the three case studies. The binary data used in the benchmarks is publicly available and does not contain any sensitive information.

A.2.2 How to access

The artifact is available at:

https://doi.org/10.5281/zenodo.15611103

A.2.3 Hardware dependencies

To properly load the Docker image, we recommend a machine with at least 8 GB of RAM and 20 GB of disk space.

A.2.4 Software dependencies

While any operating system that supports Docker can be used to run the artifact, we recommend using a Linux-based OS (e.g., Ubuntu 20.04 or later) with Docker 27.5.1 or later installed.

A.2.5 Benchmarks

The artifact includes benchmarks for three real-world binary formats:

- Bitcoin 1K: 1,000 uniformly sampled blocks (out of ~870,000 blocks at the time of writing) from the Bitcoin main chain, with about 670 MB of data.
- TLS/Tranco: Handshake traces of TLS connections from making HTTPS requests the top 100 most visited domains according to the Tranco list [7].
- PolyBenchC [2]: A canonical Wasm benchmark [6] consisting of 30 C programs compiled to Wasm.

All benchmarks are already included in the Docker image, and can be run with the provided scripts (see subsection A.4).

A.3 Set-up

A.3.1 Installation

Download the Docker image from Zenodo (https://doi.org/10.5281/zenodo.15611103) and load it using the following command:

```
docker load -i vest-image.tar.gz
```

This will create a Docker image named `vest`.

> **Note:** The image file is create from macOS with the default BSD tar, so if you are using Linux, Windows, or Windows WSL, you might need to first extract the `tar.gz` with `tar -xvzf vest-image.tar.gz`, and then load the image with `docker load -i vest-image.tar`.

A.3.2 Basic Test

To start the Docker container, run the following command:

```
docker run -it --rm vest-image
```

This will start a new container based on the `vest` image.

> **Note:** The image is built for the linux/amd64 architecture, so if you are using an Apple Silicon (M1, M2, M3, etc.) machine, you might need to run the docker image with `docker run -platform=linux/amd64 -it -rm vest-image`.

From now on, every command should be run inside the Docker container.

Once inside the container, you can run the following command to verify that VestLib and VestDSL are correctly installed, respectively:

```
cd vest
make
verification results:: 477 verified, 0 errors
```

This will compile and verify VestLib.

```
cd vest-dsl
cargo run --release -- --help
Vest: A generator for formally verified
parsers/serializers in Verus

Usage: vest-dsl [OPTIONS] <VEST_FILE>
```

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Vest delivers runtime performance on-par with (or better than) state-of-the-art, hand-written, unverified parsers and serializers.

(C2): Vest achieves verification performance that is orders of magnitude better than prior work that also produces fast and verified parsers and serializers.

A.4.2 Experiments

(E1): [2 human-minutes + 10 compute-minute + 1GB disk]: This experiment is meant to support the claim **(C1)** above.

How to: The benchmark suites for Bitcoin, TLS 1.3, and WebAssembly are respectively located in the directories `vest-dsl/bitcoin`, `vest-dsl/tls`, and `vest-dsl/wasm`. Simply run `cargo bench` from within each of these directories to see the runtime performance comparison between Vest's parsers/serializers and the hand-written, unverified parsers/serializers (i.e., Rust Bitcoin [3], Rustls [4], and `wasmparser` [5]).

Results: The the Rust Criterion benchmarking library [1] that is used to run the benchmarks will output the results in a human-readable format, including the average time taken by each parser/serializer to process a single message, as well as the standard deviation of the measurements.

(E2): [1 human-minute + 5 compute-minutes + 1GB disk]: This experiment is meant to support the claim **(C2)** above.

How to: Similar to the previous experiment, the benchmark suites for Bitcoin, TLS 1.3, and WebAssembly are respectively located in the directories `vest-dsl/bitcoin`, `vest-dsl/tls`, and `vest-dsl/wasm`. Simply run `make` from within each of these directories to verify (and time the verification time) of the parsers/serializers generated by Vest.

Results: The verification results (number of verified components and errors, if any, as well as the total verification time) will be printed to the console. All produced parsers/serializers should verify within seconds without any errors.

All results of the experiments should be comparable to the results presented in the Evaluation section (Section 7) of the paper. We do not include the verification result for EverParse [8] in this artifact, as it will take several hours to verify.

A.5 Notes on Reusability

Vest is designed to be (re)usable by developers who want to get high-assurance and performant parsers and serializers

for their binary formats. To use Vest, most developers can define their binary format in VestDSL directly, whose syntax and semantics are defined in the paper. Expert developers can also use/extend VestLib directly to build custom parsers and serializers. Though extending VestLib requires a good understanding of the Verus verifier, using VestLib should be straightforward for developers familiar with Rust.

Vest is developed in an open-source repository, where you can find the detailed documentation, examples, and tutorials on how to use it.

https://github.com/secure-foundations/vest

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/.

References

[1] Criterion.rs. https://github.com/bheisler/criterion.rs, 2024.

[2] Polybench/c. https://github.com/MatthiasJReisinger/PolyBenchC-4.2.1, 2024.

[3] Rust bitcoin. https://github.com/rust-bitcoin/rust-bitcoin, 2024.

[4] Rustls. https://github.com/rustls/rustls, 2024.

[5] wasmparser. https://github.com/bytecodealliance/wasmparser, 2024.

[6] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, and J. Bastien. Bringing the Web up to speed with WebAssembly. *SIGPLAN Not.*, 52(6):185–200, June 2017.

[7] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium*, NDSS 2019, Feb. 2019.

[8] T. Ramananandro, A. Delignat-Lavaud, C. Fournet, N. Swamy, T. Chajed, N. dim Kobeissi, and J. Protzenko. EverParse: Verified secure zero-copy parsers for authenticated message formats. In *Proceedings of the USENIX Security Symposium*, Aug. 2019.