USENIX Security '25 Artifact Appendix: PICACHV: Formally Verified Data Use Policy Enforcement for Secure Data Analytics

Haobin Hiroki Chen
Indiana University Bloomington

Hongbo Chen
Indiana University Bloomington

Mingshen Sun
Independent Researcher

Chenghong Wang
Indiana University Bloomington

XiaoFeng Wang
Indiana University Bloomington

A Artifact Appendix

A.1 Abstract

Ensuring the proper use of sensitive data in analytics under complex privacy policies is an increasingly critical challenge. Many existing approaches lack portability, verifiability, and scalability across diverse data processing frameworks. We introduce PICACHV, a novel security monitor that automatically enforces data use policies. It works on relational algebra as an abstraction for program semantics, enabling policy enforcement on query plans generated by programs during execution. This approach simplifies analysis across diverse analytical operations and supports various front-end query languages. By formalizing both data use policies and relational algebra semantics in Coq, we prove that PICACHV correctly enforces policies. PICACHV also leverages Trusted Execution Environments (TEEs) to enhance trust in runtime, providing provable policy compliance to stakeholders that the analytical tasks comply with their data use policies.

This artifact contains the Coq code, Rust implementation of PICACHV, and several tools for benchmarking PICACHV.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Since PICACHV is designed to enforce data use policies, it inherently does not pose any apparent security, privacy, or ethical concerns. Moreover, all code, scripts, and documentation are fully transparent and subject to human inspection.

A.2.2 How to access

Our artifact can be accessed on GitHub (evolving) and Zenodo (permanent archive).

- **GitHub:** An evolving version at https://github.com/hiroki-chen/picachv.
- **Zenodo:** A permanent archive: https://zenodo.org/records/14639575

Special Note: A more detailed documentation for performing artifact evaluation can be found in our repository at `docs/ae-readme.md`.

A.2.3 Hardware dependencies

We recommend conducting the evaluation on a high-end server equipped with an x86 server-grade Intel CPU (e.g., 5th Gen Intel Xeon) with Intel Trusted Domain Extension (TDX) enabled and at least 64GB of RAM. While PICACHV does not explicitly depend on TDX for performance evaluation, this hardware feature ensures end-to-end security guarantees.

A.2.4 Software dependencies

We recommend running PICACHV on Ubuntu 24.04 for optimal compatibility. Additionally, we provide a Docker image for seamless testing and evaluation.

For those who wish to build the entire artifact from scratch, the following packages must be installed beforehand.

- **System-wide dependencies:**
```
sudo apt install -y build-essential clang
python3 python3-pip python3-as-python
libgmp-dev pkg-config clang protobuf-compiler
```
- **Rust toolchain:**
```
curl -proto '=https' -tlsv1.2 -sSf
https://sh.rustup.rs | sh -s
```
- **Install `opam` to get Coq:**
```
bash -c "sh <(curl -fsSL
https://opam.ocaml.org/install.sh)"
opam init
eval $(opam env)
opam pin add coq 8.19.0
```

- **(optional) Install `mold`:**
  ```
  cd /tmp
  git clone -branch stable https://github.com/rui314/mold.git
  cd mold
  sudo ./install-build-deps.sh
  cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_CXX_COMPILER=c++ -B build
  cmake -build build -j‘nproc‘
  sudo cmake -build build -target install
  ```

- **Install python dependencies for benchmarks and data preparations.**
  ```
  pip3 install polars coloredlogs pydantic pydantic_settings pyarrow
  ```

If you are using docker, please install docker toolchains via docker's official installation guide at https://docs.docker.com/engine/install/ubuntu/

A.2.5 Benchmarks

Our benchmarks primarily require two datasets: (1) a simulated dataset generated using `TPC-H`, which is included in our artifact, and (2) case study data collected from Kaggle, available in a separate repository attached as a submodule in the main repository.

A.3 Set-up

First of all, clone our repository via `git clone --recursive https://github.com/hiroki-chen/picachv.git`

- For docker users:
  ```
  cd picachv
  docker build -f docker/Dockerfile -t picachv/picachv .
  ```
- There is currently nothing to do for users who want to build the artifact from scratch.

A.3.1 Installation

We have already introduced all the installation steps in Section A.2.4.

A.3.2 Basic Test

- **Test if Coq works:** This can be done by simply running the following script in the `picachv-proof-lib` directory of our artifact.
  `./run --allow-admitted` should yield no type errors.
- **Test if `picachv` works:**
  `cd ./benchmarks/micro/polars && cargo b -r` should yield no compilation errors.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): *The Coq formalism and security proofs are indeed mechanically checked.*
(C2): *The overall runtime overhead is practical compared to insecure baslines shown in the macrobenchmarks in Figure 15 and Table 2.*
(C3): *The runtime overhead is less relevant to the policy type but rather more impacted by the aggregation and project operators that involve policy transformation.*
(C4): *Our policy is indeed expressive enough and can capture real-world use cases as demonstrated in Section 7.3.*

A.4.2 Experiments

Preparation. In order to perform the following experiments, we need to first generate the data tables and policies needed. In our experiments, we use `tpch-dbgen` located in `benchmark`. For the convenience of testing, we provide a Python wrapper script called `prepare_data.py`.

(E1): *End-to-end latency / macrobenchmark [30 human-minutes + 30 compute-minutes + 10GB disk]:* This experiment showcases the runtime overhead of PICACHV when compared with the non-policy-enforced `tpc-h` queries. We illustrate the steps below.

How to: Generate the data and run the `tpc-h` benchmark.

Preparation: At the beginning please use the data preparation script to generate the required data at scale factor 1.0:
`cd benchmark && python3 ./prepare-data.py --scale-factor=1.0`. Then generate the mock policy files for testing.
```
cd ../tools/policy-generator
cargo run -r -- --output-path ../../data/policies/ --format parquet
```

Execution: Navigate back to `benchmark/polars-tpc` and run the following:

- Insecure baseline: `cargo run -r -- --query=[#num]`
- TPC-H with PICACHV: `cargo run -r -- --query=[#num] --policy-path ../../data/policies --enable-profiling`

Replace #num with the query number in Figure 15. (optional) After the results of all queries are collected, please open `tools/plotting/macro.ipynb` and fill in `baseline_polars` and `policy_polars` the results just obtained from the benchmark to produce Figure 15 shown in our paper.

Results: Should be able to reproduce the results in Figure 15.

(E2): *Microbenchmarks [30 human-minutes + 30 compute-minutes + 5GB disk]:* Microbenchmarks will measure 1) the cost breakdown of each relational operator (Figure 14), and 2) two specific benchmarks on the project and aggregate operators.
How to: Generate the data and run the microbenchmark.
Preparation: Similar to **E1**, please have data generator `prepare_data.py` at hand.
Execution: Due to heavy use of commands involved, please kindly refer to the `docs/AE-README.md` for more detailed instructions.
Results: This microbenchmarks should be able to reproduce Figure 14 and 16.

(E3): *Case studies [30 human-minutes + 30 compute-minutes + 5GB disk]*: This experiment shows that PICACHV can indeed support real-world use cases.
How to: Run the case studies and get the results.
Preparation: None.
Execution: Navigate to the `case_studies` repository and then run each the case study program. Due to heavy use of commands involved, similar to E2, please kindly refer to the `docs/AE-README.md` for more detailed instructions.
Results: Table 2 should be reproduced.

A.5 Notes on Reusability

We provide a brief guide on reusing our artifact for future research. The Coq formalism of relational data structures and semantics can be seamlessly integrated into other Coq projects by simply downloading and importing the relevant files. Additionally, the security proofs serve as reference proof tactics for verifying other security properties, extending beyond those discussed in the paper.

The implementation of PICACHV is a standalone Rust library designed for integration with existing data analytics frameworks. This integration requires appropriate code refactoring to invoke the corresponding PICACHV APIs. Additionally, the runtime APIs can be exposed via foreign function calls for non-Rust environments, though some manual effort may be required.

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/.