USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Let's Move2EVM

Lorenzo Benetollo, *Ca' Foscari University of Venice, University of Camerino, and Christian Doppler Laboratory Blockchain Technologies for the Internet of Things;* Andreas Lackner, *TU Wien;* Matteo Maffei and Markus Scherer, *TU Wien and Christian Doppler Laboratory Blockchain Technologies for the Internet of Things*

https://www.usenix.org/conference/usenixsecurity25/presentation/benetollo

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

USENIX Security '25 Artifact Appendix: **Let's Move2EVM**

Lorenzo Benetollo[*‡§¶], Andreas Lackner[*†], Matteo Maffei[†¶], Markus Scherer[†¶]
[†]TU Wien
[‡]Ca' Foscari University of Venice
[§]University of Camerino
[¶]Christian Doppler Laboratory Blockchain Technologies for the Internet of Things

A Artifact Appendix

A.1 Abstract

The Move programming language, designed with strong safety guarantees such as linear resource semantics and borrow-checking, has emerged as a secure and reliable choice for writing smart contracts. However, these guarantees depend on the assumption that all interacting contracts are well-formed—a condition naturally met in Move's native execution environment but not in heterogeneous or untrusted platforms like the Ethereum Virtual Machine (EVM). This work addresses the challenge of preserving Move's security guarantees when compiling to EVM.

We provide multiple artifacts to support the claims made in the paper and to facilitate data reproduction. In particular, the following artifacts are included:

- **Move-to-EVM-Compiler:** The source code of our adaptations of the original Move-to-EVM Compiler.
- **Datasets:** The Rosetta and ERC-20 Dataset used in the evaluation. Due to licensing issues, we cannot provide data from the Aptos benchmark. However, we provide scripts to build the set from publicly available sources.
- **Supplementary artifacts:** Scripts and tools to simplify running our experiments.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Our experiments do not perform any destructive operations and do not collect any private or sensitive data. To reproduce the Aptos benchmark used in the paper, we rely solely on publicly available information. Data scraping is conducted conservatively to avoid overloading the target system.

[*]Shared first authorship.

A.2.2 How to access

Our artifacts are publicly available at `https://doi.org/10.5281/zenodo.15591737`. The artifact consists of the following files:

- `aptos_dataset.zip`: A zip file containing parts of the aptos dataset and tools to generate the full dataset.
- `docker.zip`: A zip file containing additional files needed to create the Docker image.
- `Dockerfile`: The Dockerfile to create an image containing our compiler.
- `evaluation.zip`: A zip file containing tools and data to reproduce our results.
- `gas_prices.zip`: A zip file containing gas prices that we refer to in the paper.
- `lets-move-to-evm.zip`: A zip file containing the source code of our compiler.
- `paper_extended.pdf`: The extended version of the paper.
- `playground.zip`: A playground to compile Move packages in general.
- `README.md`: Additional documentation on how to run the compiler in general.

In the following descriptions, we assume that you have downloaded all files of the artifact and unzipped the (top-level) zip files mentioned above.

A.2.3 Hardware dependencies

System Requirements Artifacts can be run on both x86 and ARM architectures. We recommend 8 GB of RAM and at least 20 GB of available disk space.

A.2.4 Software dependencies

Operating System Experiments were conducted on Ubuntu 24.04. Other systems might work, but may require additional adjustments.

Docker The compiler runs inside a Docker container. All experiments were tested with Docker version 27.5.1, although newer versions will also work. To download the latest version, we refer to the official documentation: https://www.docker.com/get-started/.

Nix We use the Nix package manager for parts of our evaluation. We refer to https://nixos.org/download/ for installation instructions. To reproduce our environment, we suggest version 2.18.1 or higher.

Avoiding Nix If you have the Zstandard CLI installed and Python 3 with the `requests` package, you might still be able to run our experiments. However, please note that we have not tested this setup.

A.2.5 Benchmarks

The benchmark datasets for Rosetta and ERC-20 are included in the artifact. To retrieve the Aptos dataset, we refer to **A.3.1** for detailed instructions.

A.3 Set-up

A.3.1 Installation

Download files and install dependencies First, download the artifact from https://doi.org/10.5281/zenodo.15591737 in any directory and unzip the zip files as described before. From now on, we assume that your working directory looks as follows (assuming you have deleted unzipped zip files):

```
aptos_benchmark
├── docker
├── evaluation
├── gas_prices
├── lets-move-to-evm
├── playground
├── Dockerfile
├── paper_extended.pdf
└── README.md
```

Preparing benchmarks For the Aptos benchmark, due to licensing issues, we cannot directly provide the source files. Anyhow, we provide scripts to download and patch the files:

```
cd aptos_benchmark
nix-shell
./build_eval.bash
```

Note that two modules (`offer` and `universal_config`) will be skipped. This is as expected and relates to compiler limitations described in the paper. The results of this script are copied to the `evaluation` folder. After that, you can leave the nix shell again by pressing `CTRL+d`.

Building the Docker image The Docker image contains the compiler and all its dependencies. Building the Docker container can last up to 30 minutes. Note that we expect you to execute the next line in the root folder of your downloaded artifact (where the `Dockerfile` is located)

```
docker build -t mv2evm .
```

This should build a new Docker image containing the compiler. You can verify whether everything was successful by running `docker image ls`. The list of images should contain `mv2evm`.

A.3.2 Basic Test

After successfully running the previous steps, you can execute the following command to see if everything worked correctly:

```
./evaluation/attach.bash <COMPILER> <DOCKERIMG>
```

For `<DOCKERIMG>` use `mv2evm` as created before. For `<COMPILER>`, test it for each of the values `IRM`, `ORIGINAL`, `SOL`. Executing the script `attach` should open a shell (inside the created Docker container). Note that it might take up to one minute until the Docker container is started.

Inside the new shell you should be able to execute the command `move --help` which should return a list of available command-line-options. **You can leave the Docker container now by typing** `CTRL+d`.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1) From Section 2.3: The original compiler is vulnerable to the bugs referred to as C1, C2, and C4 in the paper (names unfortunately collide with the Major Claims in this section). This is proven by (E1).

(C2) From Section 4.2 §ERC-20: When using Move in an *unidiomatic way* (as proposed by the original compiler), gas costs nearly double with respect to the ERC-20 version of Solidity (C2.1). Using Move as intended leads to an overhead of at most 42% (C2.2) when using the (insecure) original compiler. Compiled with our compiler, we get an overhead of at most 50% (C2.3), where only 15% are caused by our changes in the compiler. This can be proven by (E2).

(C3) From Section 4.2 §Rosetta: For the Rosetta benchmark, all claimed contracts (see Figure 7) can be compiled by our compiler (C3.0). On average, the overhead

compared to the original compiler is less than 49,000 (C3.1). Compared to Solidity, it is less than 98,000. This is proven by (E3).

(C4): From Section 4.2 §Aptos: For the Aptos benchmark, all claimed contracts (see Figure 8) can be compiled by our compiler (C4.0). The overhead compared to the original compiler is on average around 12,000 of gas (C4.1). This is proven by (E4).

A.4.2 Experiments

NOTE: In this section, the working directory will be the `evaluation` directory!

(E0): Execute all tests [30 human-minutes + 10 compute-minutes]: This experiment is the basis for most of the other experiments. It must be repeated for each of the three compilers. We use `<COMPILER>` to indicate the values `IRM`, `SOL`, and `ORIGINAL`.
Preparation: Attach to the Docker container by typing `./attach.bash <COMPILER> mv2evm`.
Execution: Execute the script `./evaluate.bash` that should be placed in the current working directory of the just-opened shell inside the Docker container.
Results: The script executes test cases. There must not be any failing test cases (failing test cases would be noted at the end of the output). Also, typing `ls ./results` must yield a non-empty list of `csv` files. Warnings are expected and can be ignored.

(E1): Vulnerability Check [<10 human-minutes + <10 compute-minutes]:
Preparation: Attach to the Docker container by typing `./attach.bash ORIGINAL mv2evm`
Execution: Execute the script `./vul_check.bash` that should be placed in the current working directory of the just-opened shell inside the Docker container.
Results: The script should execute three tests. All of them must be passed (indicated by `3 passing`). Compiler warnings are expected and can be ignored.

(E2): ERC-20 [30 human-minutes + <10 compute-minutes]:
Preparation: First execute (E0). Then leave the Docker container (`ctrl+d`). You should now be in folder `evaluation`.
Execution: Execute the script `./e2.bash`.
Results: We expect the following output: 88% for (C2.1); 28% for (C2.2); 36% for (C2.3); and 20% for (C2.4). We note that numbers are slightly **better** than those presented in the paper since the Solidity version performs worse in our test environment. Other numbers are unchanged (compare output with Figure 6 of the paper).

(E3): Rosetta [30 human-minutes + <10 compute-minutes]:
Preparation: First execute (E0) if not done previously. Then leave the Docker container (`ctrl+d`). You should now be in folder `evaluation`.
Execution: Execute the script `./e3.bash`.
Results: We expect the following output: 35,482 for (C3.1); and 89,942 for (C3.2). Regarding C3.0, the script should output results for all contracts listed in Figure 7. We note that the numbers for IRM slightly differ from the paper due to fixes. However, in case of increasing costs, the increase should be smaller than 3% (see last column printed by the script).

(E4): Aptos [30 human-minutes + <10 compute-minutes]:
Preparation: First execute (E0) if not done previously. Then leave the Docker container (`ctrl+d`). You should now be in folder `evaluation`.
Execution: Execute the script `./e4.bash`.
Results: We expect a value of 12,174 for C4.1. Regarding C4.0, the script should output results for all contracts listed in Figure 8. Numbers should match those of the paper up to the nearest thousand (as represented in the paper).

A.5 Notes on Reusability

For the reproducibility of the data mentioned in the paper, we only need a small part of the Aptos dataset. However, in the `README.md` of the artifact, we provide detailed instructions on how to retrieve the whole dataset.

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/.