USENIX Security '26 Artifact Appendix: Wormholes in the File System: Understanding the Misunderstanding of Symlinks

Yongheng Liu
Fudan University

Lei Zhang *
Fudan University

Yuhang Zhao
Fudan University

Yuzhou He
Fudan University

A Artifact Appendix

This artifact appendix is intended to serve as a self-contained document for evaluating our artifact. It provides a structured roadmap for reproducing the findings in our paper "Wormholes in the File System: Understanding the Misunderstanding of Symlinks".

A.1 Abstract

The artifact includes the source code, datasets, scripts, results and instructions needed to reproduce the main experimental results reported in the paper. It has two main parts. `PathResolution` part reproduces our measurements of file type identification and path normalization interfaces across ten programming languages. `PathTraversalFinding` part contains our archive-testing framework, which constructs ZIP, TAR, and 7z proof-of-concept archives for the bypass and arbitrary-write strategies described in the paper, and evaluates them against archive extraction libraries and tools.

A.2 Description & Requirements

This section lists all the necessity to recreate the same experimental setup we have used to run our artifact.

A.2.1 Security, privacy, and ethical concerns

Conducting the artifact does not raise any security, privacy, or ethical concerns. All experiments are conducted within isolated Docker containers. The vulnerability detection tool generates crafted archive files for testing purposes only and does not perform any destructive operations on the host.

A.2.2 How to access

The artifact is available at our Zenodo repository: https://doi.org/10.5281/zenodo.20325581.

A.2.3 Hardware dependencies

No special hardware is required. Our testing environment is built on an x86_64 Linux server.

A.2.4 Software dependencies

The artifact requires Docker to be installed on the Linux system. All other dependencies are encapsulated within the provided Docker images. No additional software installation is needed on the host.

A.2.5 Benchmarks

The benchmarks are included in the artifact. For path resolution, we consider two categories of scenarios: file type identification and path normalization. The artifact includes the symlink path cases used to support our experimental analysis. During execution, the artifact dynamically generates the required directory structures for these cases. In addition, we provide testing wrappers for 35 file type identification interfaces and 34 path normalization interfaces across ten programming languages. For path validation, the artifact includes wrappers and dependency scripts for 85 archive extraction projects, together with scripts that dynamically generate the test archives used in our experiment.

A.3 Set-up

This section includes all the installation and configuration steps required to prepare the environment to be used for the evaluation of our artifact.

A.3.1 Installation

Download and extract the archived artifact, then enter the repository root:

```
cd SymlinkMisunderstanding
```

Build the path-resolution image:

```
cd PathResolution
docker build -t symlink-pathresolution .
```

Build the path-traversal image:

```
cd ../PathTraversalFinding
docker build -t ptr-check .
```

*Corresponding author

A.3.2 Basic Test

After building the Docker image, run a basic functionality test:

```
docker run --rm symlink-pathresolution bash -c "echo OK"
docker run --rm ptr-check bash -c "echo OK"
```

Expected output: `OK`. This confirms that the Docker image is correctly built and the language runtime environments are functional.

A.4 Evaluation workflow

This section includes the operational steps and experiments required to evaluate the functionality of our artifact and validate the key results and claims of our paper.

A.4.1 Major Claims

Our major claims include the following:

(C1): *Path resolution behavior experiments.* We identify four aspects of confusion in file type identification and five distinct normalization strategies across ten languages.

(C2): *Path traversal vulnerability detection.* Our tool can automatically detect path traversal vulnerabilities in archive extraction projects. Through evaluation of 85 projects, we uncover 16 arbitrary file write vulnerabilities.

A.4.2 Experiments

The corresponding experimental steps are as follows:

(E1): *Path resolution behavior experiments [30 human-minutes + 1 compute-hours]*
Reproduce the path resolution inconsistency analysis across 10 programming languages.
Preparation: Enter `PathResolution/` directory. Mount `PathResolution` using an absolute path:

```
docker run -it \
    -v "$PWD:/workspace/PathResolution" \
    symlink-pathresolution bash
```

Execution: Inside the container, run the file type identification tests:

```
cd /workspace/PathResolution/FileTypeIdentification
./run/run_all.sh
```

Then run the path normalization tests:

```
cd /workspace/PathResolution/PathNormalization
./run_all.sh
python3 classify.py
```

Results: CSV files are generated under `FileTypeIdentification/Results/` and `PathNormalization/Results/`, with one file per language. The classification script outputs a strategy table (S1–S4, Abs-Rel) and writes `Results/_strategy_classification.csv`. Results should match the pre-computed outputs in `ExperimentResults/`.

(E2): *Path traversal vulnerability detection [30 human-minutes + 2 compute-hours]*
Reproduce the path traversal vulnerability detection on 85 archive extraction projects.
Preparation: Enter `PathTraversalFinding/` directory. Mount PathTraversalCheck using an absolute path:

```
docker run -it -v \
 "$PWD/PathTraversalCheck:/opt/ptr/PathTraversalCheck" \
 ptr-check
```

Prepare dataset dependencies inside the container:

```
cd /opt/ptr/PathTraversalCheck
bash datasets/setup_all.sh
```

Set up the Python environment:

```
cd /opt/ptr/PathTraversalCheck
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

Execution: Run the detection tool:

```
python main.py
```

Results: The tool outputs a summary of vulnerability detection results. The expected output should match:

```
summary:
  total: 85
  arbitrary write: 16
  bypass arbitrary read: 2
  arbitrary read: 22
  no issue: 45
```

Detailed per-project results are written to `PathTraversalCheck/out/`, including generated PoC archive files under the `seeds/` subdirectories.

A.5 Notes on Reusability

The artifact can be extended to test additional archive extraction libraries. To add a new target, users can add an entry to `test.json` and provide the corresponding wrapper implementation. The bypass and write strategies implemented in the tool are modular and can be adapted to evaluate other path validation scenarios beyond archive extraction.

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/usenixsec2026/.