USENIX Security '25 Artifact Appendix: Exploring and Exploiting the Resource Isolation Attack Surface of WebAssembly Containers

Zhaofeng Yu[1], Dongyang Zhan[1]*, Lin Ye[1], Haining Yu[1], Hongli Zhang[1], Zhihong Tian[2]
[1]Harbin Institute of Technology, [2]Guangzhou University
yuzhaofeng@stu.hit.edu.cn
{zhandy, hityelin, yuhaining, zhanghongli}@hit.edu.cn
tianzhihong@gzhu.edu.cn

A Artifact Appendix

A.1 Abstract

This artifact provides a comprehensive research toolset for analyzing WebAssembly runtimes and evaluating the resource exhaustion exploits proposed in our paper. It consists of two major components: a static analysis framework and a suite of automated exploit scripts.

The static analysis framework performs static analysis of WASI/WASIX interfaces to identify external API calls and potential syscall invocations, along with their relevant parameters. To support reproducibility, we provide both a Docker image with all dependencies pre-installed and standalone automation scripts for non-containerized environments.

The exploit part of the artifact demonstrates how crafted WebAssembly instances can exhaust host resources via legitimate runtime interfaces. It includes test cases and automation scripts to reproduce our evaluation results and measure the impact of each exploit instruction.

A README file is provided, which contains detailed setup and usage instructions. This artifact is released for academic use only. Any malicious application is strictly prohibited.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The static analysis component of our artifact does not pose any security risks. During exploit evaluation, some test cases may temporarily affect system performance, such as I/O throughput and network bandwidth. For network-related experiments, users need to specify an IP and port, and run tools like `ncat` and `iperf3` on a separate host to receive traffic (see README for details). Our artifact does not modify or collect any user data. It does not perform any destructive actions.

* Corresponding author: Dongyang Zhan (Email: zhandy@hit.edu.cn)

A.2.2 How to access

Our artifact is publicly available on Zenodo. Please visit the following DOI to access the latest version: https://doi.org/10.5281/zenodo.16594863.

The archive contains all necessary files, including analysis tools, exploit scripts, Docker images, and documentation. Please refer to the included `README` file for detailed setup and usage instructions.

A.2.3 Hardware dependencies

The artifact does not require any specialized hardware. However, we recommend a system with at least 16 GB of RAM (ideally 32GB) and a reasonably powerful CPU to ensure smooth analysis and reasonable execution time.

A.2.4 Software dependencies

To begin with, we strongly recommend reading the provided `README` file, which offers detailed instructions for environment setup and dependency installation. As an initial step, please ensure that `p7zip-full` is installed in order to extract the artifact package.

If using the Docker image, the analysis framework requires only Docker to be installed. However, for users who prefer to run the framework locally without Docker, it is necessary to manually install LLVM (version 17.x) and a nightly version of Rust that is compatible with the selected LLVM version. In addition, several other system dependencies are required, including `curl`, `cmake`, `gcc`, `g++`, `python3`, `python3-dev`, `zlib1g`, `make`, `binutils`, `bzip2`, `zip`, `xz-utils`, and `git`.

For evaluating the exploits, a separate set of dependencies must be installed. These include `python3`, `python3-pip`, `dstat`, `ent`, `fio`, `iperf3`, and `sysstat`. Additionally, the Python package `pandas` is required and can be installed using `pip`.

A.2.5 Benchmarks

None

A.3 Set-up

A.3.1 Installation

For cases where the provided Docker image is used to test the static analysis framework, please follow the steps described in Listing 1.

```
# Step 1: Install Docker
# Step 2: Unpack the archive to retrieve our
    provided Docker image.
sudo apt update
sudo apt install p7zip-full
7z x artifact.image.7z.001
# Step 3: Load the Docker image
sudo docker load -i artifact.image
```

Listing 1: Environment setup for evaluating the static analysis framework with Docker.

For cases where the static analysis framework is tested in a non-containerized environment, please follow the steps described in Listing 2.

```
# Step 1: Install dependencies
sudo apt --yes update && sudo apt install --yes
    cmake gcc g++ \
              python3 python3-dev zlib1g make
                  binutils bzip2 \
              zip xz-utils git
git clone https://github.com/llvm/llvm-project.git
mkdir ./build && cd ./build/
cmake -S ../llvm-project/llvm/ -B ./ -G "Unix␣
    Makefiles" \
              -DCMAKE_BUILD_TYPE=Release \
              -DLLVM_ENABLE_PROJECTS="clang;clang-
                  tools-extra;lldb;lld" \
              -DLLVM_ENABLE_RUNTIMES="libcxx;
                  libcxxabi;compiler-rt;libunwind" \
make -j 8
sudo make install
sudo apt install --yes curl
curl --proto '=https' --tlsv1.2 -sSf https://sh.
    rustup.rs | sh
rustup toolchain install nightly-2024-02-01
rustup default nightly-2024-02-01-x86_64-unknown-
    linux-gnu
rustup component add rust-src --toolchain nightly
    -2024-02-01-x86_64-unknown-linux-gnu
# Step 2: Unpack the archive to retrieve our
    artifact.
sudo apt update
sudo apt install p7zip-full
7z x wasm_artifacts.7z.001
```

Listing 2: Environment setup for evaluating the static analysis framework without Docker.

To evaluate the exploit strategies proposed in our paper, please follow the steps described in Listing 3 to set up the environment.

```
# Step 1: Install dependencies
sudo apt install python3 python3-pip
pip install pandas
curl https://wasmtime.dev/install.sh -sSf | bash
curl https://get.wasmer.io -sSfL | sh
sudo apt install dstat ent fio iperf3 sysstat
# Step 2: Unpack the archive to retrieve our
    artifact.
sudo apt update
sudo apt install p7zip-full
7z x wasm_artifacts.7z.001
```

Listing 3: Environment setup for evaluating the proposed exploit strategies.

A.3.2 Basic Test

To test our static analysis framework, please follow the instructions described in Listing 4 to run it.

```
# After loading the provided Docker image, you can
    run the container to automatically demonstrate
    the workflow of the static analysis framework.
sudo docker run -it --rm artifacts:latest
# After installing the dependencies and extracting
    the provided artifact, navigate to the directory
     and run the automated script.
cd wasm_artifacts
./start.sh
```

Listing 4: Running the static analysis framework via Docker or automated script.

To evaluate the exploit strategies proposed in our paper, please follow the instructions described in Listing 5.

```
# After installing the dependencies and extracting
    the provided artifact, navigate to the directory
     and run the automated script.
cd wasm_artifacts/exploit
./exploit.sh
# Note: If you need to evaluate network-related
    strategies, each time you run the script (
    including the first), you must provide two
    parameters. You can run ncat and iperf3 on
    another machine and then supply their listening
    IP addresses and ports to the script.
./exploit.sh <send_traffic_ip:port> <iperf_server_ip:
    port>
```

Listing 5: Evaluating our proposed exploit strategies via automated script.

A.4 Evaluation workflow

A.4.1 Major Claims

Our artifact consists of two components: the static analysis framework and our proposed exploit strategies. To evaluate the artifact, one can examine the outputs produced by both components, as described below.

(C1): *In experiment (E1), the static analysis framework is executed using the provided Docker image (recommended) or the automated script, producing a JSON file that contains key external APIs or syscalls involved in the runtime interfaces, along with some possible parameters. Table 1 in Section 3.4 of our paper illustrates part of the contents of this JSON file.*

(C2): *In experiment (E2), the provided automated script is used to measure the system's baseline performance and evaluate the impact of each exploit strategy on system performance. The output results are saved in* `wasm_artifacts/exploit/data/result`*. In the WebAssembly container, our strategies can degrade system performance in various aspects, such as I/O, CPU usage, and network bandwidth. Several tables in Section 5 of our paper present these results.*

A.4.2 Experiments

(E1): *[Run the static analysis framework] [About 1 hour.]: By running our static analysis framework via the Docker image or the automated script, the framework will automatically build the project, extract the IR, perform analysis, and finally output a JSON file containing the key external APIs and syscalls involved in the runtime interfaces, along with some possible parameters.* ***Note that this process takes a considerable amount of time. Therefore, we have included a GIF in the README to demonstrate the procedure.***

Preparation: *If you want to run the static analysis framework using the Docker image (recommended), follow the steps in Listing 1. To run it using the automated script, follow the installation steps in Listing 2.*

Execution: *The basic commands are listed in Listing 4, which by default analyze the Wasmtime runtime. The source code of the runtime is already included in both the Docker image and the artifact, so no additional download is required. If you wish to analyze Wasmer instead, an extra parameter must be provided when running the container or script:* `/wasm_artifacts/wasm_runtimes/wasmer/` *for the container, and* `./wasm_runtimes/wasmer/` *for the script. Note that analyzing Wasmer can take significantly more than one hour due to its large codebase. It is also recommended to ensure that the system has more than 30GB of available memory to avoid analysis failures due to memory exhaustion.*

Results: *After the Docker container or automated script finishes running, the contents of the JSON file will be displayed using* `cat`*. In the container, the JSON file is saved at* `/wasm_artifacts/result`*, while the script saves it to* `./wasm_artifacts/result`*.*

(E2): *[Optional Name] [About 4 hours]: By running the provided automated script multiple times, the system baseline performance and the system performance during the execution of the exploit strategies can be collected.* ***We have also included a GIF in the README to demonstrate the procedure.***

Preparation: *To evaluate the impact of the exploit strategies, first install the dependencies by following the instructions in Listing 3.*

Execution: *The basic commands are listed in Listing 5, which will first measure the system baseline performance and then evaluate the impact of the first exploit strategy. By running the script multiple times, you can continuously evaluate the next exploit strategy.*

Results: *When the script is run but produces no output, it means all exploit strategies have been evaluated. All collected data can be found in the* `wasm_artifacts/exploit/data/result` *directory.*

A.5 Notes on Reusability

All scripts and source code for our static analysis framework and exploit tools are included in the artifact. If needed, new features can be added by simply modifying the scripts and source code. Additionally, the analysis-related source code can be directly integrated into your own project and used by calling the provided class interfaces, requiring only a proper LLVM installation.

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/.