Enhancing the security of **WebAssembly** runtimes using **Linux Security Modules**

Hardening WASI using Landlock LSM

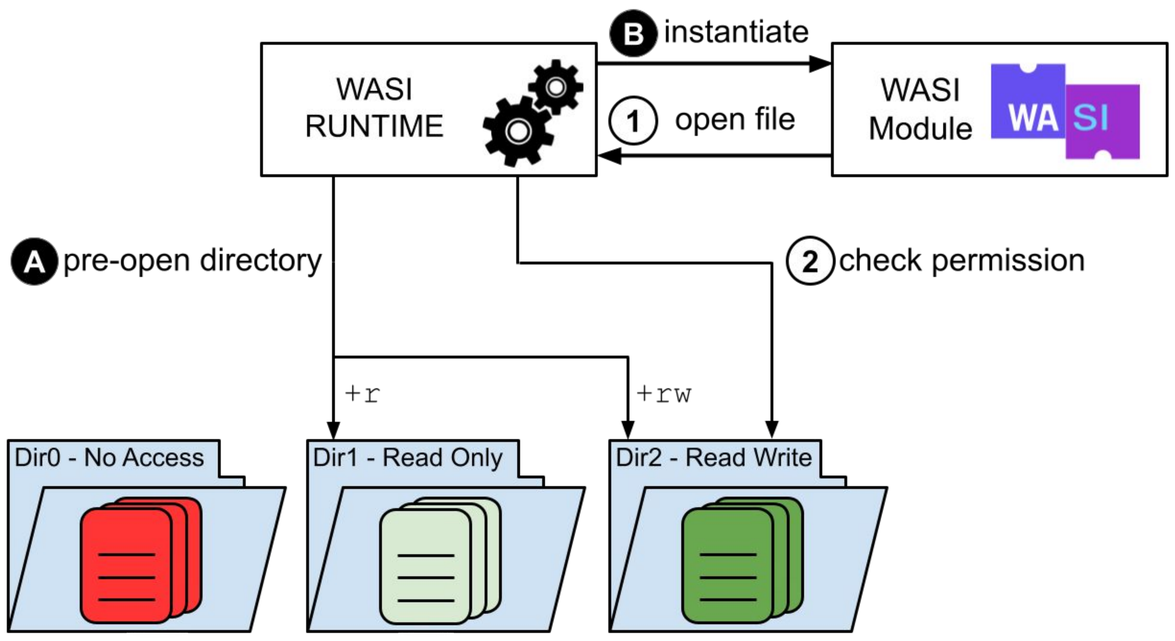
Marco Abbadini, Dario Facchinetti, **Gianluca Oldani**, Stefano Paraboschi, Matthew Rossi

BACKGROUND

- **WebAssembly runtimes** enable the execution of WebAssembly programs outside of the browser
- the new use-case comes with additional requirements and, as a result, to access the underlying system the **WebAssembly System Interface (WASI)** has been standardized

PROBLEM STATEMENT

The **WASI filesystem sandbox** is implemented using WASI-libc, which exposes syscalls on top of a libpreopen-like layer



This method has the following limitations:

- WASI-compliant runtimes must provide their own implementation of syscalls wrappers
- **no protection** when the **runtime** is affected by a **vulnerability**
- **limited** access control **granularity** (directory-level)

RISKS INTRODUCED BY THE CURRENT DESIGN

- directory-level permissions force the developer to separate confidential and non-confidential files into distinct folders; when this structure is not adopted, **confidential data may be leaked** by buggy or untrusted modules
- runtimes may behave differently due to corner cases in the preopen logic, leading to **ambiguity** in **sandbox definition**

METHODOLOGY

- constrain access to the filesystem through the **Landlock LSM**
- evaluate the performance overhead w.r.t. current solutions on benign and malicious WASI modules

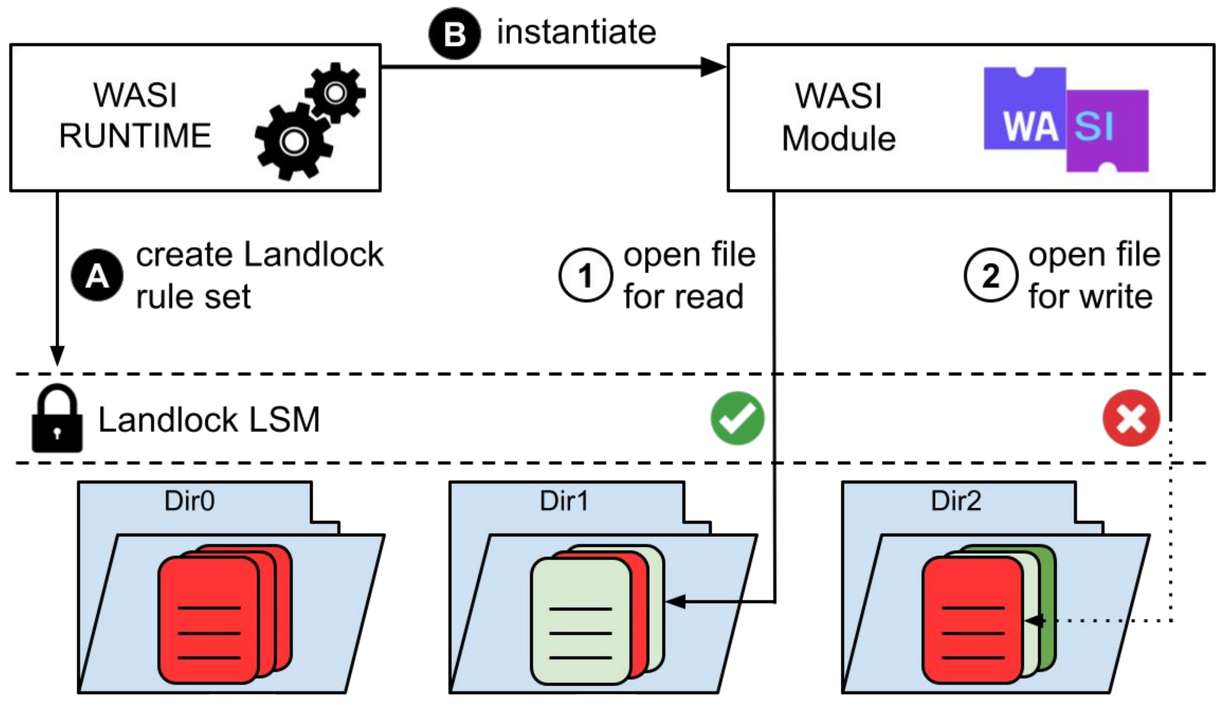
IDEA

Replace the runtime preopen logic with a call to Landlock. In the example below the host .pem keys can be accessed read-only.

```
let mut state_builder = WasiState::new("genrsa-wasi");
let state = state_builder
                .args(&["genrsa", "-out", "keys/key.pem", "2048"])
                .preopen(|p| p.path("keys/key.pem").read(false).write(true).create(true))?
                .build()?;
```

Avoid storing permissions in a global state, use instead the kernel API to set permissions at process level.

```
let status = Ruleset::new()
                    .handle_access(AccessFs::from_all(ABI::V1))?.create()?
                    .add_rules(rules_from_vec(&vec![keys/key.pem],
                                                ACCESS_FS_ROUGHLY_WRITE))?
                    .restrict_self()?;
```



ADVANTAGES

- access control enforced by the Kernel
- no need for a custom libc implementation
- same behavior across different runtimes
- **file-level granularity**, instead of directory-based access
- preliminary evaluation shows limited overhead compared to current implementation (**~1% overhead**)

GET A COPY OF THE POSTER!