Enhancing WASI Sandboxes

Restricting WebAssembly Runtimes with Linux Security Modules

Michele Beretta
Relatore: Stefano Paraboschi

Corso di Laurea Magistrale in Ingegneria Informatica
Università degli Studi di Bergamo

30 settembre 2022, AA 2021/2022

JavaScript

JavaScript (JS) è il linguaggio standard (de facto) per sviluppare applicazioni sul web.

Per piccole applicazioni è più che sufficiente, ma possono sorgere dei problemi:

1. parsing lento per via della rappresentazione testuale
2. tipizzazione dinamica e poche possibilità di ottimizzazione
3. vulnerabilità, tra cui XSS e CSRF

UNIVERSITÀ DEGLI STUDI DI BERGAMO

WebAssembly

WebAssembly (WASM) è un *formato binario* di istruzioni nato nel 2015, supportato da quasi tutti i browser.

Obiettivi principali:

- *sicurezza*
- *velocità*
- *portabilità*
- *compattezza*

WASM risolve alcuni problemi di JS introducendo:

1. parsing più veloce
2. type system statico
3. memoria dati e codice separate

UNIVERSITÀ DEGLI STUDI DI BERGAMO

Runtime WASM e WASI

Nonostante WASM sia nato per il web, i suoi punti di forza lo rendono adatto per il sandboxing di binari anche offline.

Nascono quindi le *runtime WASM* che permettono di eseguire WASM direttamente su sistemi operativi.

Vista la necessità di accedere a funzionalità del sistema, si ha la *WebAssembly System Interface* (WASI), un'interfaccia modulare per accedere a file system, variabili d'ambiente, ed altre risorse.

UNIVERSITÀ DEGLI STUDI DI BERGAMO

Limiti delle runtime WASM

L'uso da riga di comando delle runtime WASM presenta alcuni svantaggi:

- è possibile dare accesso solo a cartelle intere
- non è possibile specificare che permessi ha il programma WASM

Questo lavoro di tesi si pone l'**obiettivo** di risolvere queste limitazioni tramite le funzionalità di sandboxing offerte dal kernel Linux, in particolare Linux Security Modules.

UNIVERSITÀ DEGLI STUDI DI BERGAMO

Linux Security Modules – LSM

Framework di access control per il kernel Linux.

Permette di effettuare controlli di sicurezza e di descrivere tramite delle regole cosa un processo può fare.

I due framework usati sono:

- *Landlock*, un framework per access control non privilegiato
- *eBPF*, una macchina virtuale che offre sandboxing in contesto privilegiato

UNIVERSITÀ DEGLI STUDI DI BERGAMO

Restrizione con Landlock

Nuovo progetto scritto in *Rust* che usa *Wasmtime*. Permette di definire vari permessi legati al file system.

```
./rust-wasm-landlock copy-file.wasm
    --dir .
    --fs-allow input.txt:R
    --fs-allow output.txt:W
```

UNIVERSITÀ DEGLI STUDI DI BERGAMO

Vantaggi e svantaggi

Vantaggi:

- semplice per l'utente
- non necessita di permessi speciali

Svantaggi:

- scope ristretto
- assenza di eccezioni

UNIVERSITÀ DEGLI STUDI DI BERGAMO

Restrizione con eBPF

Si usa *BPFContain*, un wrapper per eBPF scritto in Rust. Si hanno:

- un *server* che applica le varie restrizioni
- un *client* che gestisce l'eseguibile

Le restrizioni sono definite con una *policy*.

```
name: copy-file
cmd: ./wasmtime copy-file.wasm --dir .

allow:
- fs: {pathname: ./wasmtime, access: rxm}
- fs: {pathname: ./input.txt, access: r}
- fs: {pathname: ./output.txt, access: w}
```

Policy

```
$ bpfcontain daemon fg
$ bpfcontain run copy-file.yml
```

Comandi

UNIVERSITÀ DEGLI STUDI DI BERGAMO

Vantaggi e svantaggi

Vantaggi:

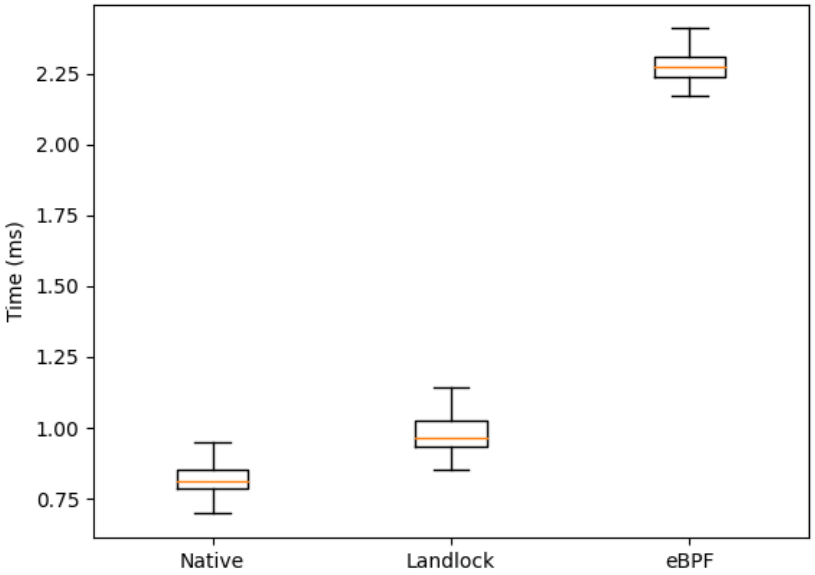
- permessi precisi e vari
- definizione di eccezioni
- non si applica solo al file system

Svantaggi:

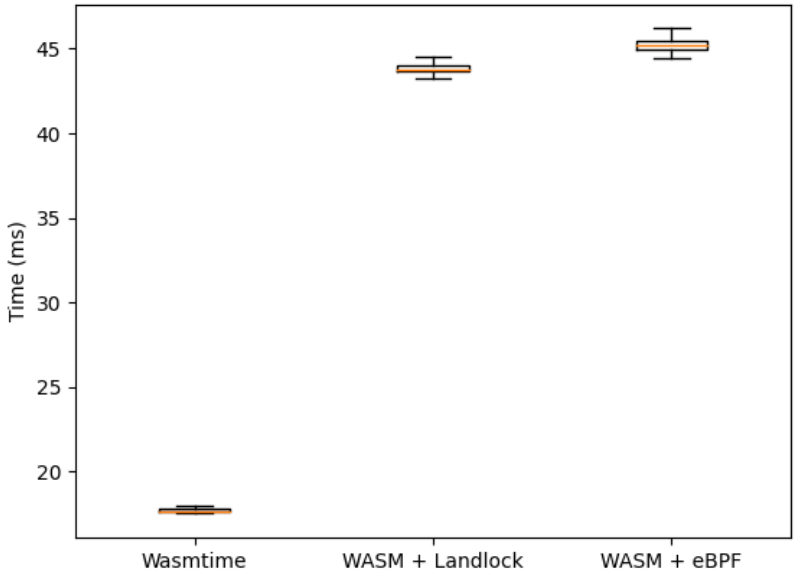
- complesso
- necessita di permessi speciali

UNIVERSITÀ
DEGLI STUDI
DI BERGAMO

Ordinamento di 10000 numeri



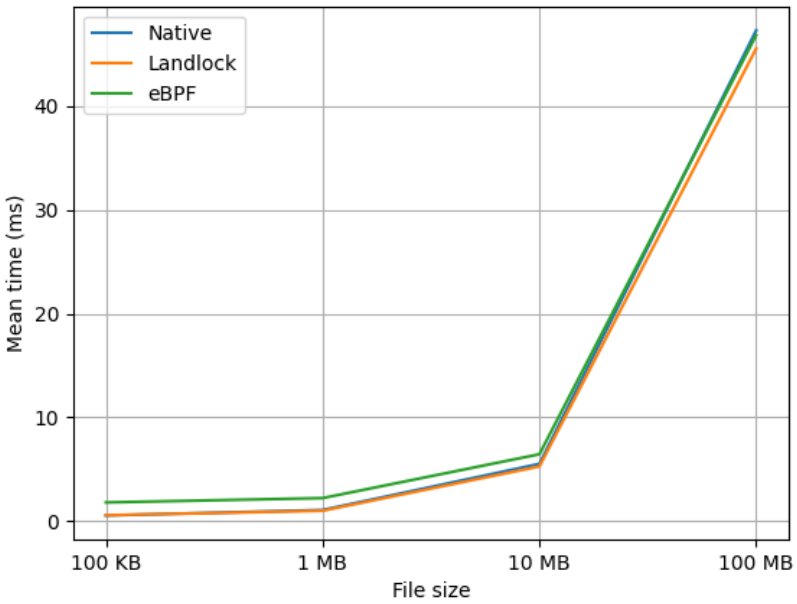
Nativo



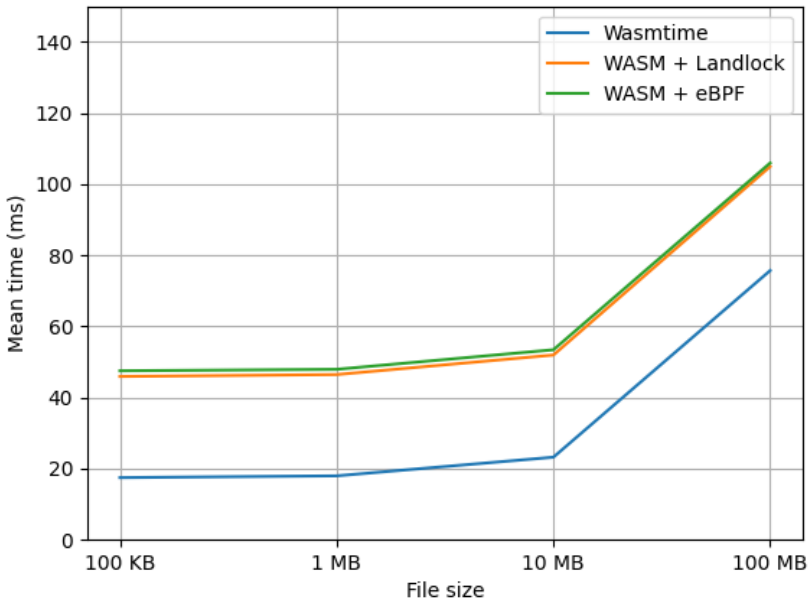
WASM

UNIVERSITÀ DEGLI STUDI DI BERGAMO

Lettura di file



Nativo



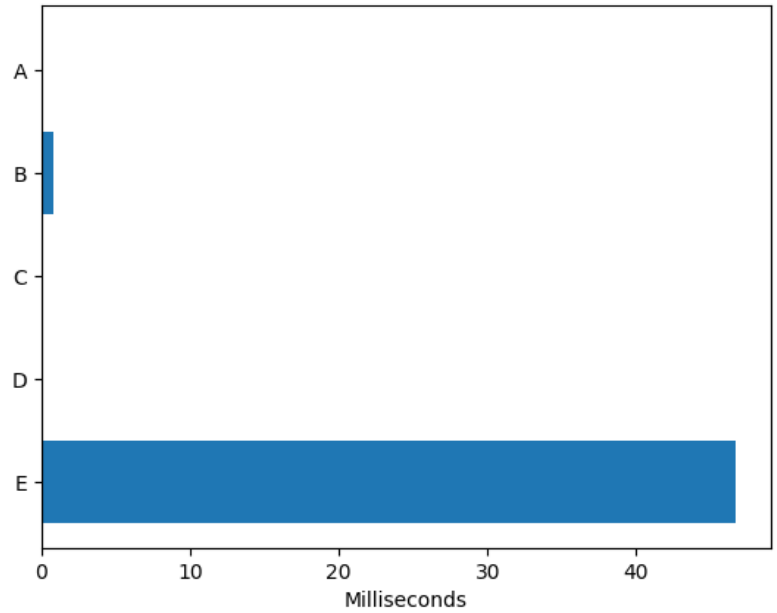
WASM

UNIVERSITÀ DEGLI STUDI DI BERGAMO

Analisi interna della runtime WASM

In ordine, le sezioni sono:

A. parsing di argomenti
B. inizializzazione del modulo WASM
C. apertura di cartelle
D. applicazione di Landlock
E. interpretazione del modulo WASM

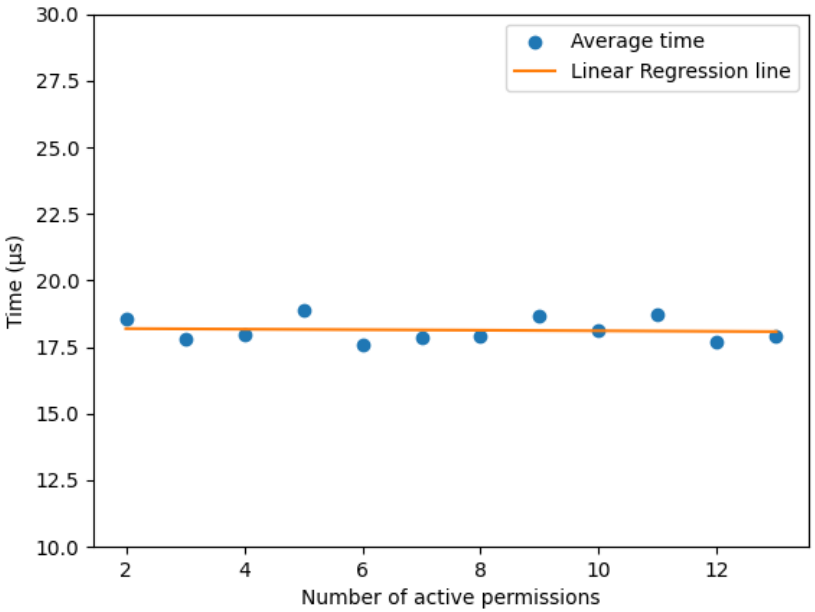


L'interpretazione del modulo WASM causa il rallentamento.

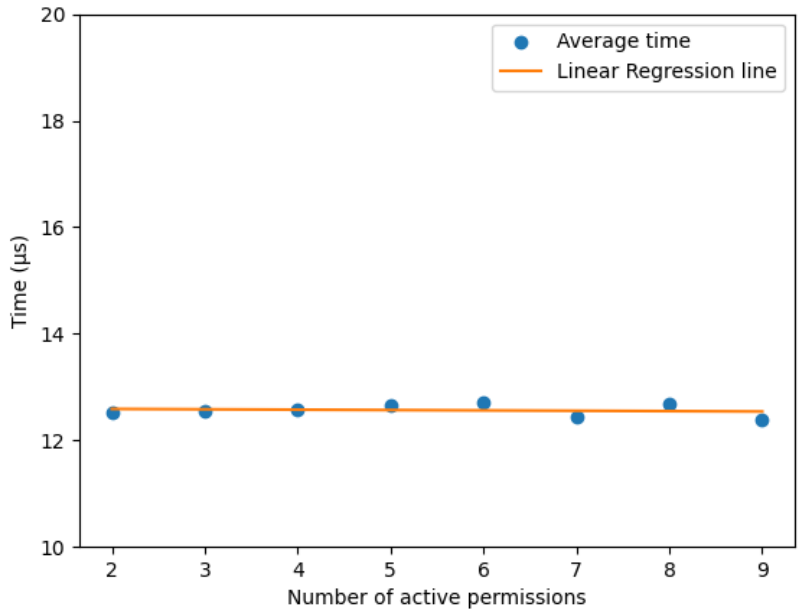
UNIVERSITÀ DEGLI STUDI DI BERGAMO

Dipendenza dal numero di permessi

Come la restrizione iniziale del processo varia in base al numero di permessi su un singolo percorso.



Landlock

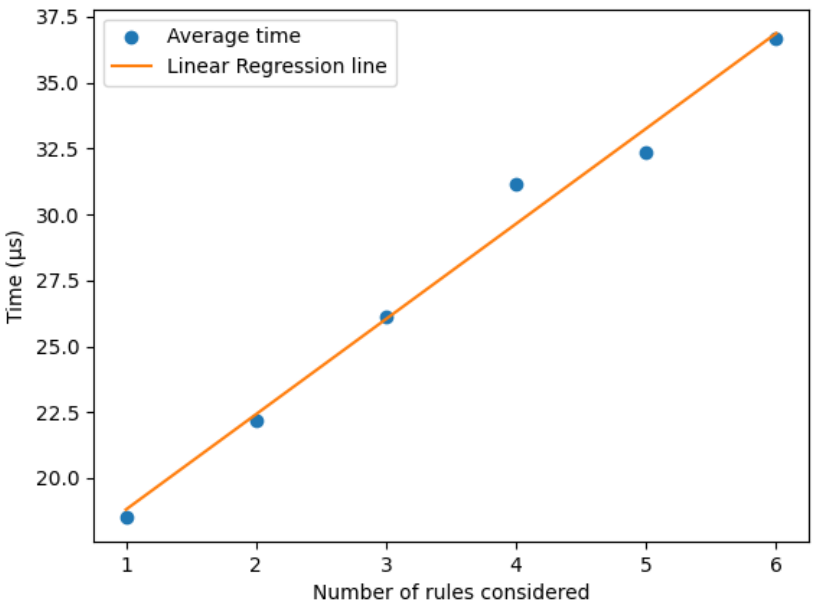


eBPF

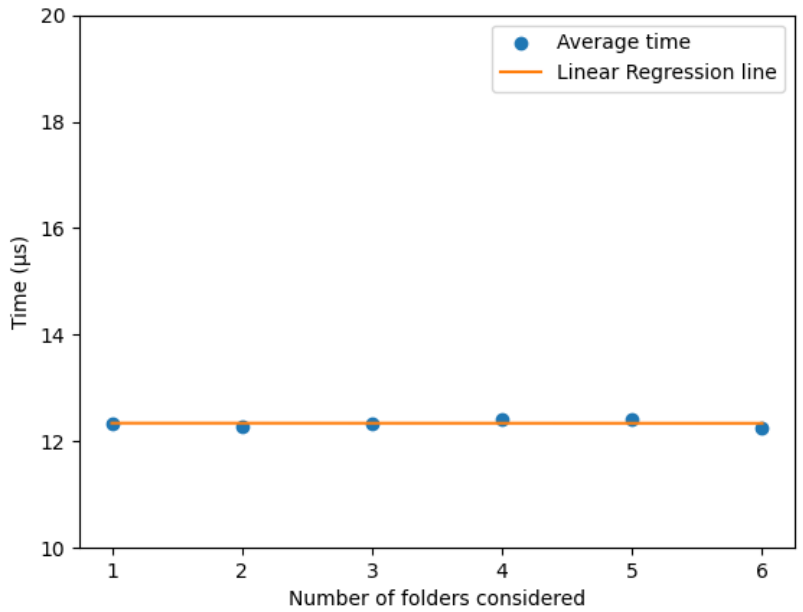
UNIVERSITÀ DEGLI STUDI DI BERGAMO

Dipendenza dal numero di regole

Come la restrizione iniziale del processo varia in base al numero di cartelle accessibili (ogni regola dà accesso ad una sola cartella).



Landlock



eBPF

UNIVERSITÀ DEGLI STUDI DI BERGAMO

Conclusioni

Si conclude che:

- WASM è ancora lontano da performance native
- l'interpretazione di WASM fa da collo di bottiglia
- l'uso di sandbox non pregiudica fortemente le performance
- Landlock è più "facile" di eBPF, ma meno completo

Possibili sviluppi futuri:

- creazione di un framework per access control da integrare con le runtime
- integrazione di Landlock con la runtime, ma restringendo la portabilità

UNIVERSITÀ
DEGLI STUDI
DI BERGAMO

Grazie dell'attenzione

Domande?

`https://github.com/micheleberetta98/rust-wasm-landlock`