rkt-io

A Direct I/O Stack for Shielded Execution

https://github.com/Mic92/rkt-io

Jörg Thalheim
Harshavardhan Unnibhavi, Pramod Bhatotia

Christian Priebe
Peter Pietzuch

ACM EuroSys 2021
Security in the untrusted infrastructure

How do we ensure application security in untrusted cloud environments?
Trusted computing

Hardware-assisted trusted execution environments (TEEs)

Offered by major cloud providers
I/O in TEEs: “Strawman design”

Worldswitch is 5x slower than syscalls
Current I/O mechanisms: “Switchless design”

Switchless design avoid world-switches on the I/O path
Limitations of switchless designs

● **Performance**
  ○ **OS bottlenecks**: The OS is still on the I/O path
  ○ **I/O threads**: Needs dedicated I/O threads, require tuning to find optimal number of threads
  ○ **Data copies**: Additional data copies between TEE ↔ IO threads ↔ OS

● **Compatibility**
  ○ LibOS-based approaches often only provide a subset of Linux ABI
rkt-io: A Direct I/O library for TEEs

rkt-io combines a library OS with direct I/O libraries

**Application**

**System call interface**

**LibOS**

**Direct I/O interface**

**Host memory**

**Trusted TEE**

**Compatibility:** Full Linux ABI

**Performance:** OS-bypass & less copies

**Untrusted host**

**NIC**

**SSD**
Direct I/O in TEEs

Direct I/O improves IO performance significantly
Design
Design principles

1. I/O stack interface
2. I/O event handling
3. I/O stack partitioning
I/O interface: Host OS independence
1. Host OS independent I/O stack

**Network:**
- DPDK (Data Plane Development Kit)
- Direct access to fast network interface
- No TCP/IP stack
- No socket

**Storage:**
- SPDK (Storage Performance Development Kit)
- Direct access to NVMe devices from userspace
- No filesystem
- No file abstraction

How to maintain compatibility with existing APIs?
rkt-io: Host OS independent I/O stack

- **Library OS (LKL)**
- **Direct I/O libraries**
- **Hardware**

**TEE**
- Musl libc / Linux ABI
- syscalls

- **Linux network stack**
- **Linux filesystems**

**Driver interface**
- **DPDK**
- **SPDK**

**Direct hardware access**
- NIC
- SSD

- Full compatibility for existing applications
- Trusted TEE
- Host-OS bypass

- Untrusted host
2

I/O event handling: Polling-based approach
2. I/O event handling

Interrupts cause world switches in TEE!
2. Rkt-io’s polling driver (for NIC)

rkt-io uses device polling to handle the I/O events
3

I/O stack partitioning:
Control and data path partitioning
3. I/O stack partitioning for TEEs

Application

write()

LibOS with direct I/O

Return to app, app reuses buffer

Protected memory

NVMe queues

First copy!

Second copy!

Sq

Cq

How to avoid the second copy?
One-copy network/block device driver

SKB = Linux’s internal buffer socket

Before

TCP/IP stack

Send

SKB

Receive

SKB

Copy

Untrusted memory

After

TCP/IP stack

Send

SKB

Receive

SKB

Copy

No copy!

Untrusted memory

Page cache/socket buffer are written to unencrypted memory
Usage
Usage

$ rkt-io-run /app-disk.img /usr/bin/redis-server --bind 10.0.1.1

- Applications are packaged as encrypted, signed filesystem images
- Transparent substitution of musl libc at runtime
- I/O path is (optionally) encrypted:
  - Full disk encryption with cryptsetup
  - Layer-3 VPN with wireguard

rkt-io enables applications to easily build with a commodity package manager
Evaluation
Evaluation: Benchmarks and applications

- Synthetic benchmarks:
  - Storage (fio) and network (iPerf)

- Real-world applications:
  - SQLite (Speedtest), nginx (wrk), Redis (YCSB), MySQL (Sysbench)

- Baselines
  - **Non-secure**: Native Linux
  - **Secure**: SCON (host OS) & SGX-LKL (library OS)
Network stack: iPerf

Rkt-io provide high iPerf network throughput due to NIC offloading
Storage stack: fio (random read-write)

Disk throughput ahead to other frameworks but behind native (smaller page cache)
Evaluation: MySQL (sysbench)

Both SGX-LKL and rkt-io are faster than native.
## Analysis of futexes in MySQL

<table>
<thead>
<tr>
<th>Top 5</th>
<th>Syscall</th>
<th>Count</th>
<th>Time (μs)</th>
<th>Total (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>futex</td>
<td>64</td>
<td>4.20e+07</td>
<td>69.4</td>
</tr>
<tr>
<td>#2</td>
<td>read</td>
<td>24728</td>
<td>9.40e+06</td>
<td>15.5</td>
</tr>
<tr>
<td>#3</td>
<td>select</td>
<td>9</td>
<td>8.99e+06</td>
<td>14.8</td>
</tr>
<tr>
<td>#4</td>
<td>fsync</td>
<td>436</td>
<td>6.03e+04</td>
<td>0.1</td>
</tr>
<tr>
<td>#5</td>
<td>write</td>
<td>8243</td>
<td>3.48e+04</td>
<td>0.06</td>
</tr>
</tbody>
</table>

Breakdown of Top-5 syscalls in MySQL native execution

**MySQL performance benefits from LibOS futexes / context switching**
Summary

- Current SGX-implementation are not designed for high-performance I/O
  - **OS bottlenecks**: The OS is still on the I/O path
  - **I/O threads**: Require tuning to find optimal number of I/O threads
  - **Data copies**: Additional data copies between TEE ↔ IO threads ↔ OS

- **rkt-io provides**
  - Transparent and fast access to the I/O devices
  - Linux ABI-compatibility for applications in TEEs

**Performant + Secure + Transparent**

**Try it out!**

[https://github.com/Mic92/rkt-io](https://github.com/Mic92/rkt-io)