Enarx

Protection for data in use

Mike Bursell
Office of the CTO

Nathaniel McCallum
Sr. Principal Engineer

https://enarx.io
Trusted Execution Environments
Trusted Execution Environments

TEE is a protected area within the host, for execution of sensitive workloads.
Trusted Execution Environments

TEE provides:
- Memory Confidentiality
- Integrity Protection
- General compute
- HWRNG

TEE is a protected area within the host, for execution of sensitive workloads.
TEE provides:
- Memory Confidentiality
- Integrity Protection
- General compute
- HWRNG

Q. “But how do I know that it’s a valid TEE?”
Q. “But how do I know that it’s a valid TEE?”
A. Attestation

TEE provides:
- Memory Confidentiality
- Integrity Protection
- General compute
- HWRNG

Trusted Execution Summary
TEE provides:

- Memory Confidentiality
- Integrity Protection
- General compute
- HWRNG

Attestation includes:

- Diffie-Hellman Public Key
- Hardware Root of Trust
- TEE Measurement
TEE provides:
- Memory Confidentiality
- Integrity Protection
- General compute
- HWRNG

Attestation includes:
- Diffie-Hellman Public Key
- Hardware Root of Trust
- TEE Measurement

Trusted Execution Summary
Introducing Enarx
Enarx Principles

1. We don’t trust the host owner
2. We don’t trust the host software
3. We don’t trust the host users
4. We don’t trust the host hardware
   a. ... but we’ll make an exception for CPU + firmware
Enarx Design Principles

1. Minimal Trusted Computing Base
2. Minimum trust relationships
3. Deployment-time portability
4. Network stack outside TCB
5. Security at rest, in transit and in use
6. Auditability
7. Open source
8. Open standards
9. Memory safety
10. No backdoors
Enarx Architecture

Application

Language Bindings (libc, etc.)

WASI

WebAssembly

Process-Based Keep

SGX
Sanctum

VM-Based Keep

SEV
PEF
MKTME

W3C standards
Enarx is a Development Deployment Framework

Choose Your Language / Tools → Develop Application → Compile to WebAssembly

Choose Host → Instance Configuration
Just enough legacy support to enable trivial application portability.
Homogeneity to enable radical deployment-time portability.
No interfaces which accidentally leak data to the host.
Bridges process-based and VM-based TEE models.
No operating system to manage.
Process flow
Overview (AMD example)

“Server”

Host

- AMD firmware

Secure VM

- Code runs

Attestation handshake

“Client”

Tenant

- Code + data delivery (encrypted)
Enarx architectural components

Host

Application

Enarx runtime

Enarx host agent

CPU + firmware

Client

Enarx client agent

CLI

Orchestrator (e.g. OpenShift/k8s, OpenStack)

Code + Data (Encrypted)

Client/host agent comms

Attestation

1, 5

6

2, 4

1, 5

3, 7

Keep
Enarx attestation process diagram

1. Request workload placement
2. Request Keep
3. Create Keep, load Enarx runtime
4. Measurement of Keep + Enarx runtime
5. OK/not-OK
6. Code + Data (encrypted)
7. Load Code + Data into Keep

CLI / Orchestrator

Enarx client agent

Enarx host agent

CPU/firmware

Enarx Keep
Enarx Status
Current Status

1. SEV: Fully attested demo w/ custom assembly.
   a. Ketuvim: KVM library with SEV support
2. SGX: Fully attested demo w/ data delivery.
3. PEF: Ongoing discussions with POWER team.
4. WASM/WASI: Demo with some basic WASI functions.
We Need Your Help!

Website: https://enarx.io

Code: https://github.com/enarx

Master plan: https://github.com/enarx/enarx/issues/1

License: Apache 2.0

Language: Rust
Questions?

https://enarx.io