Ultraviolet: A Code Integrity Model for Minimal Container Hosts

Linux Plumbers Conference 2025

James Morris
jamorris@linux.microsoft.com

In This Talk

- Brief overview of Ultraviolet project, a WIP.
- Seeking early feedback from community & collaboration.

What is UV?

- An architectural model.
- A reference implementation.

Scope

- Minimal container host for hyperscale infrastructure.

Architecture – High Level

- Extends community image-based Linux efforts.
- Supports generalized launch-time code integrity.
- Technology-agnostic model.

Threat Environment

- Sophisticated, well-funded attackers.
- High value, and large scale of assets to be protected.
- Global-scale deployment (millions of nodes), high overall complexity.

Policy-Driven

- Declare what's on a node & how it's configured.
- Enforce integrity and security properties of the node.
- Measure & attest boot process, configuration, and code launch.
- Alerting.

Code Integrity – Working Definition

- *At launch*, code is:
    - Of known provenance.
    - Unmodified.
    - Correctly authorized.

Code Integrity Goals

- Prevent unauthorized or modified code from launching at OS level:
  - Extend trusted boot to full system
  - Compiled binaries, scripts, kernel modules, loadable firmware, etc.
- Enforce configuration integrity (code behavior).
- *Control* over JIT & other dynamic code.

General Requirements for Code Integrity

- Requires trusted boot process and hw root of trust.
- Need to protect integrity of the underlying system (e.g. via MAC).
- Protection against runtime attacks.
- Assume breach: constrain attacker, prevent persistence, ensure detection and preserve forensics.

UV Model: Assumed OS Components

- Base OS (standard, minimal, image-based, A/B updates etc.)
- Container runtime
- Containerized workloads
  - Control plane
  - Network plane
  - Data plane
  - App plane (etc.)

So, what is missing?

- A generally accepted form of **state separation**.
    - For full coverage code integrity, we need to separate read-only and writable data in a standardized manner for all components.
- Controlled sharing: isolation is easy, sharing is hard.
- Security abstraction: fully utilize container boundaries, signed verity images, etc.
- Declarative system-wide policies.

Modest Proposal

- A broadly agreed model for state separation, and where all code above the base OS & container runtime:
  - Is containerized, in a conforming container model
  - Is well-defined in functional scope (e.g. control-plane)
  - Is strongly isolated from other containers and layers
  - Conforms to state-separation of read/write data
  - Shares data with other components via standardized policy-controlled channels (e.g. IPC, files)

What does this look like?

- Reference implementation (uv-linux), early WIP.
- Based on learnings from existing projects like Azure Boost and OS Guard, but more generalized and closer to upstream.
- Image-based configuration of Fedora Rawhide, utilizing ParticleOS.
- Starts with latest UAPI Group model, latest systemd, & kernel. Extends with IPE (Integrity Policy Enforcement), state-separation, updated SELinux model, OBS support, script integrity.

Discussion

Resources

- Projects
  - UV Linux Project
  - ParticleOS
  - UAPI Group
  - Integrity Policy Enforcment (IPE)

- Conference Talks
  - Azure Boost by Luca Boccassi at All Systems Go (ASG)
  - A Security Model for systemd by Lennart Poettering at ASG
  - OS Guard by Sriram Nambakam at SCaLE