Zen Crypted Operating System

Технічне завдання

на розробку та впровадження операційної системи OS.1 з ядром Linux та бібліотеками BSD

На основі Технічних Вимог

Згідно з Постановою КМУ № 205 від 21.02.2025

Формалізованих за стандартом ISO/IEC/IEEE 29148:2018

OS.1:
A Monolithic, Zero-Trust Architecture for Hardened Edge and Desktop Systems

BitEdits Engineering Group

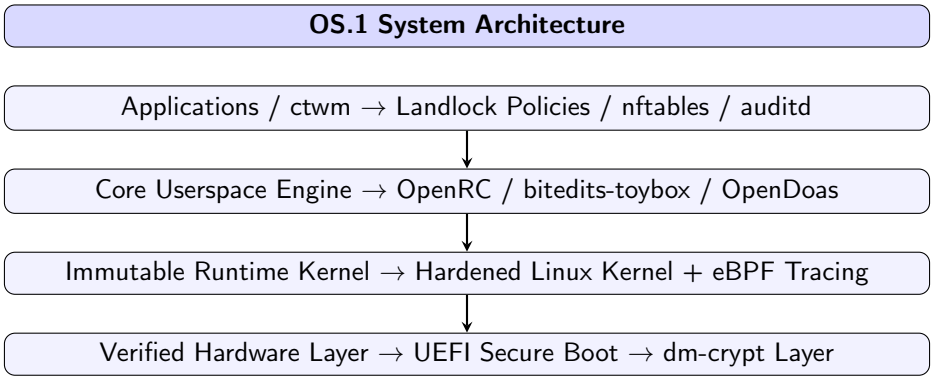
1 серпня 2026 р.

Зміст

1. Introduction

OS.1 (internally designated as **zencrypted/os**) is a monolithic, zero-trust operating infrastructure designed for high-assurance computing. Modern operating systems suffer from bloated attack surfaces, architectural coupling with system management daemons (`systemd`), and complex boot chains vulnerable to physical and logical tampering.

By synthesizing an ultra-lightweight independent rootfs framework, an immutable Unified Kernel Image (UKI), declarative hardware sandboxing, and strict privilege isolation, OS.1 eliminates legacy attack vectors. The operating system boots directly from UEFI NVRAM into an encrypted memory-mapped environment, bypassing traditional bootloaders, and executing a sandboxed userspace governed by `musl-libc`, `toybox`, and the Landlock Linux Security Module (LSM).



Architectural Pillars

The design of OS.1 relies on four uncompromising principles:

- **Cryptographic Immutability:** The entire base operating system kernel, configuration matrix, and initial execution files reside in a single, cryptographically signed binary.
- **Attack Surface Minimization:** Traditional distributions deploy up to several gigabytes of software. OS.1 strips the base rootfs down to its absolute mathematical minimum via custom forks (`bitedits/*`).
- **Process Isolation by Default:** Applications lack system-wide visibility. Access to the filesystem, network socket layers, and hardware inputs must be explicitly granted via declarative kernel policies.
- **State Separation:** The operating system treats runtime state as volatile. Persistent changes are restricted to dedicated, encrypted data blocks.

Kernel Architecture & Security Profiles

The operating system runs an optimized downstream derivation of the Linux Hardened kernel, prioritizing compile-time exploit mitigation over raw benchmark optimizations. The kernel configuration enforces:

- Strict Page Table Isolation (`PAGE_TABLE_ISOLATION=y`).
- Kernel Stack Randomization on system calls (`RANDOMIZE_KSTACK_OFFSET=y`).
- Complete memory wiping on allocation/deallocation boundary conditions (`INIT_ON_ALLOC_DEFAULT_ON=y`).

2. Zen Crypted Operating System

2.1. Secure Boot (tpm2)

OS.1 eliminates legacy bootloaders (like GRUB) which are inherently vulnerable to tampering. Instead, the firmware directly executes an immutable Unified Kernel Image (UKI) via UEFI Secure Boot. This mechanism handles runtime decryption and initialization seamlessly.

- **Mechanism** — UEFI, signed UKI, direct kernel loading, dm-crypt (initramfs)

2.2. Limine Bootloader (limine)

Limine serves as the primary bootloader, ensuring a modern, strictly compliant UEFI boot sequence before transitioning to the Unified Kernel Image.

2.3. TianoCore EDK II (edk2)

The firmware foundation relies on TianoCore EDK II, delivering a verifiable and trusted execution environment for UEFI Secure Boot operations.

2.3.1. Unified Kernel Image (UKI) Construction

The system builds a monolithic `zencrypted-os.efi` file containing:

1. An official UEFI boot stub (`systemd-stub` stripped of systemd execution logic).
2. A highly tailored, hardened Linux kernel.
3. The system kernel command line (hardcoded at compile-time to prevent boot parameter injection).
4. A minimal, specialized stage-1 `initramfs`.

2.3.2. Cryptographic Verification

During machine activation, the hardware firmware evaluates the UKI against custom Platform Keys (`PK`/`KEK`/`db`) enrolled in the motherboard NVRAM. If verification succeeds, control hands off directly to the kernel execution thread.

2.3.3. Early-Stage Disk Decryption (dm-crypt)

The immutable `initramfs` instantiates a minimal storage driver stack to invoke Linux kernel cryptography routines.

- The primary system partition is validated and decrypted using `cryptsetup` with `dm-crypt` (AES-XTS-PLAIN64).
- Storage headers are detached or randomized to obscure partition bounds.
- Once authentication passes, the root filesystem mounts as a read-only device, executing a `switch_root` loop into the main operating workspace.

2.4. Root File-system

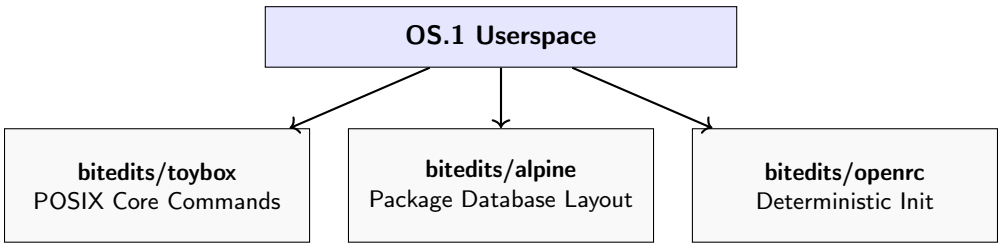
At the core of the user environment is an ultra-minimalist root filesystem derived from Alpine Linux. By managing a dedicated downstream fork, we strictly control and minimize the surface area of the base distribution.

2.5. C Library (`musl`)

An implementation of the standard library functionality described in the ISO C and POSIX standards, plus common extensions, built on top of the Linux system calls API. MUSLC is used in NuttX, Alpine, Chimera, and OS.1.

2.6. Berkley Userland (`chimerautils`)

All vital system utilities are provided by Alpine Toybox or Chimeratils BSD Userland. This establishes a hard breakpoint that separates the core system from monolithic dependencies while retaining absolute POSIX compliance.



2.7. POSIX Engine (`toybox`)

All vital system tools (`ls`, `ps`, `cat`, `chmod`) are provided via Rob Landley's `toybox`. This deployment yields:

- A clean breakpoint separating the system from bloated GNU software dependencies.
- Strict compliance with POSIX runtime directives.
- Symmetrical performance enhancements when interacting with memory operations over the `musl` API.

2.8. Power Tools (`acpi`)

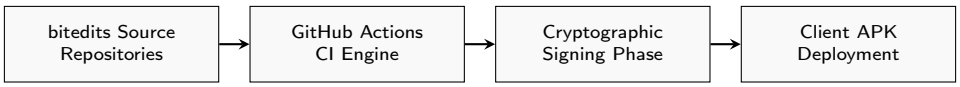
Hardware power states and performance governors are managed directly through native logic, eschewing heavy system daemons in favor of predictable, direct tuning mechanisms. Tooling relies on `acpi` and `cpupower`.

2.9. Time Server (`chrony`)

Accurate, cryptographic synchronization is paramount to securing a zero-trust architecture. Chrony serves as an isolated NTP client ensuring that system timestamps remain rigorously validated against network tampering. Chrony is used in Alpine, Chimera and OS.1.

2.10. Package Management (apk)

Software lifecycle integrity is maintained via Alpine Package Keeper (APKv3). By utilizing deterministic builds and local offline signatures, we ensure a highly defensive supply chain architecture.



1. **Isolated Source Compilation:** Custom `APKBUILD` files fetch package source trees directly from validated `bitedits/*` endpoints.
2. **Deterministic Compilation:** Packages compile within automated, isolated pipelines generating reproducible hash check validations.
3. **Cryptographic Asset Signing:** Package outputs are packaged using Alpine's native Package Keeper (`apk-tools`) format and signed with an offline master key pair.
4. **Local Repository Integration:** The rootfs image builder installs signed `.apk` payloads directly into the OS structure, refusing any package lacking verified, local cryptographic signatures.

2.11. Auditing (audit-userspace)

System call profiling and anomaly detection are captured dynamically by a localized, minimal auditing layer, entirely bypassing bloated centralized logging paradigms.

System behavior analysis relies on non-invasive inspection loops rather than intrusive hooking infrastructures. System call profiling is verified by `bitedits/audit-userspace`, capturing unauthorized privilege transition vectors and immediate file write signals.

2.12. Tracing (bpftrace)

Advanced, non-invasive system observability is delivered natively through eBPF. Security policies monitor kernel tracepoints dynamically without requiring structural alterations or intrusive kernel hooks.

Leveraging `bpftrace` patched directly for `musl-libc` interfaces, security daemons monitor kernel tracepoints dynamically without requiring run-time structural alterations.

2.13. Network Firewall (nftables)

The network boundary is fiercely guarded by a strict default-drop nftables stack. Routing policies are verified at load, ensuring only explicitly permitted, encrypted traffic traverses the network sockets.

The firewall stack loads directly within the kernel image definition, adopting a strict **Default-Drop** posture across all interface hooks:

```
[frame=single, label=nftables Core Configurations, breaklines=true, fontsize=\small]
table inet filter {chain input {type filter hook input priority filter;
policy drop;ct state established,related acceptiif "lo" accept}chain forward {type filter hook forward
priority filter; policy drop;}chain output {type filter hook output priority filter; policy drop;
udp dport 123 accept # Allowed: chrony NTP Synctcp dport 443 accept # Allowed: Outbound Encrypted TLS}}
```

2.14. Mandatory Access Control (landlock)

Moving beyond convoluted policy frameworks, OS.1 deploys thread-level unprivileged sandboxing via Landlock. Processes must securely restrict their own filesystem access through strict declarative profiles.

Unlike legacy MAC systems (SELinux/AppArmor) which add complex policy layers and complex tooling, OS.1 implements process sandboxing natively:

- **Mechanism:** Utilizing unprivileged sandboxing features, processes restrict their own filesystem actions and access paths.
- **Configuration:** The userspace engine utilizes `landlockconfig` to ingest declarative TOML rule structures at application initialization.

```
[frame=single, label=Example System Profile: /etc/landlock/ctwm.toml, breaklines=true, fontsize=\small]
[sandbox]
handled_access_fs = ["execute", "read_file", "read_dir", "write_file"]

[[rules.fs]]
path = "/usr/bin/ctwm"
access = ["execute", "read_file"]

[[rules.fs]]
path = "/home/zencrypted/.ctwmrc"
access = ["read_file"]
```

2.15. Init Supervisor (openrc)

Process initialization follows a strictly deterministic dependency graph. OpenRC eliminates parallel background bloat by mapping service dependencies cleanly and predictably during boot orchestration.

- **No Parallel Background Bloat:** Service dependencies map cleanly during build orchestration.
- **Predictable Execution Blocks:** Services run as unprivileged isolations where feasible, logging securely to stateless `/dev/log` pipes audited by `auditd`.

2.16. User Privileges (doas)

Privilege transitions are governed by OpenDoas. This minimalist framework enforces strict execution constraints with a footprint of less than 5% compared to legacy sudo, entirely preventing cross-shell context persistence.

For administrative access loops, standard `sudo` is banned. The system relies on `bitedits/OpenDoas`, enforcing:

- A code footprint less than 5% of legacy `sudo`.
- Zero context persistence between separate shell instances.
- Strict execution constraints mapped explicitly per user profile.

2.17. Seat Devices (seatd)

Input routing and hardware mediation are controlled by `seatd`. It securely permits graphical workspaces to access shared devices without elevating permissions to complex display manager suites.

To operate a graphical workspace without a resource-heavy or complex display manager, OS.1 links input routing through `seatd`.

- Grants application engines access to shared hardware interfaces (such as graphics cards or inputs) without requiring elevated `root` permissions.
- Avoids any dependency on large user tracking packages, maintaining a tiny security perimeter for input tracking.

2.18. Window Manager (ctwm)

The visual desktop layout is anchored by Claude's Tab Window Manager (`ctwm`). By bypassing complex message brokers and notification loops, ctwm drastically reduces the rendering attack surface, providing a highly productive, ultra-light interface.

- **Efficiency:** It runs inside an extremely lightweight system profile, bypassing modern desktop notification loops and complex message broker backplanes.
- **Security:** Minimizes the visual display and rendering attack surface against local interface sniffing vectors.

2.19. GStreamer (gstreamer)

Handling complex multimedia pipelines is delegated to GStreamer. Running within strictly isolated contexts, it ensures robust audio and video processing while preventing codec-level exploits from compromising the broader system. GStreamer and PipeWire are supported in Chimera and OS.1.

2.20. Erlang (erlang)

For resilient, fault-tolerant network services and daemons, OS.1 embraces Erlang. Its battle-tested actor model and extreme concurrency provide an uncompromising foundation for highly available distributed architectures.

2.21. Elixir (elixir)

Building atop the Erlang Virtual Machine, Elixir introduces modern developer tooling and extensibility. It accelerates the development of secure, scalable system services while inheriting the raw reliability of the underlying VM.

2.22. Comparative Matrix

Табл. 1: System Parameter Architectural Baseline

Security Parameter	Standard Enterprise Linux	OS.1
Boot Mechanism	UEFI → GRUB → Kernel → Initramfs	UEFI → Signed Monolithic UKI
Boot Surface Tampering	Vulnerable (Unsigned configs/modules)	Impossible (Breaks Secure Boot Signature)
Core Utilities	GNU Coreutils (∼150MB, Complex)	`bitedits/toybox` (<5MB, Minimalist)
Init Framework	Complex asynchronous initialization	Monolithic, simple OpenRC system
Default Sandbox Engine	Process-wide (SELinux/AppArmor)	Thread-level declarative Landlock
Privilege Transition	Legacy Sudo Engine (Highly complex)	OpenDoas Framework (Audited, Minimal)

3. Conclusion

OS.1 proves that modern hardware systems can be decoupled from complex, legacy distribution frameworks. By consolidating the runtime layer inside an immutable Unified Kernel Image and enforcing strict sandboxing across minimal userspace forks, OS.1 delivers an uncompromised, zero-trust infrastructure optimized for high-assurance desktop layouts and isolated edge computation nodes.