Churchill: Continuous Runtime Integrity for High-Assurance Linux Workloads

Technical White Paper

Westgate Data Science · July 2026

Abstract

High-assurance Linux workloads such as settlement brokers, regulatory data planes, and signing and key-handling services face a class of threats that exist after deployment and during execution: debugger attachment, code injection, library hijacking, privilege escalation, memory tampering, and silent binary substitution. These attacks are invisible to perimeter controls and slow for observe-and-alert EDR, which detects, ships telemetry off-host, and waits for a decision while the attacker acts.

Churchill closes this gap with a fail-closed, dual-sentinel runtime integrity architecture. Enforcement is graduated by design. The common case is a synchronous denial, in which an unauthorized execution, mount, debugger attach, memory write, or network destination is refused at the kernel boundary while the protected application keeps serving, and the attempt is recorded in a tamper-evident evidence chain. Termination is reserved for the one case denial cannot cover: verified compromise of the running workload itself. There it is immediate, irreversible, and paired with persistent lockdown and forensic evidence preservation. No version of a protected application deploys at all until a client-assigned approval quorum has cryptographically signed the exact change, with keys Churchill never holds.

This paper describes the architecture, protection layers, governance model, and operational characteristics of the system for technical evaluators, security architects, and audit audiences.

---

1. The Runtime Blind Spot

An attacker with user-space access on a host can typically manipulate a running process without tripping file-integrity or network-based alarms:

- **Debugger attachment** enables live inspection and modification of process memory, extraction of cryptographic keys, and control-flow bypass.
- **Code injection** via `LD_PRELOAD`, GOT/PLT hooking, or memory-mapped executable regions redirects program behavior without touching on-disk binaries.
- **Privilege escalation** through capability manipulation, namespace migration, or credential changes converts constrained access into full control.
- **Binary substitution** replaces authorized executables with malicious variants, surviving integrity checks that run only at load time.
- **Filesystem shadowing** uses bind mounts, overlayfs, or `pivot_root` to interpose a malicious view of trusted paths without modifying the underlying inodes.

- **Memory tampering** through `/proc/<pid>/mem` writes or `process_vm_writev` plants shellcode directly in a running process.
- **Living off the land** requires no new code at all: data leaves the host through binaries that are already present and authorized.

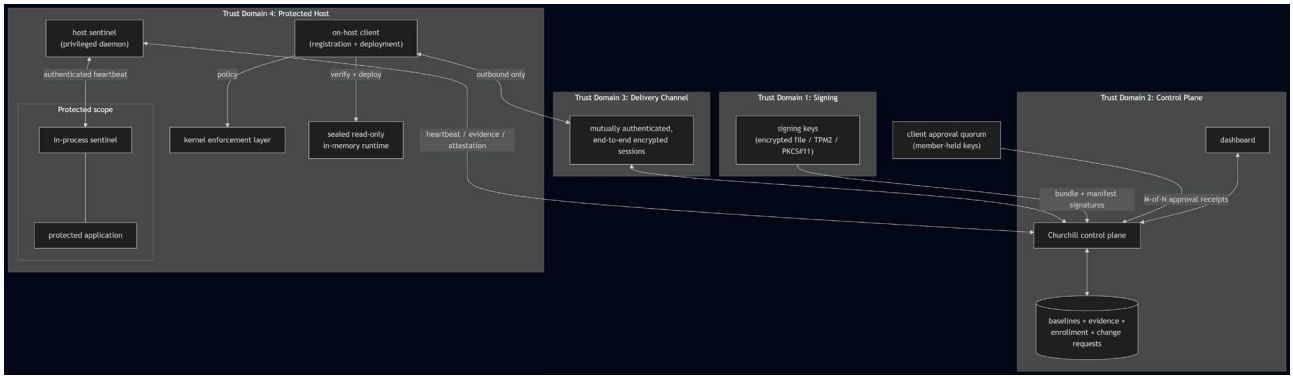
These techniques share a property: they occur inside boundaries that traditional defenses assume are safe. Churchill is designed to operate inside that perimeter, continuously validating that an application remains unmodified, unattached, unprivileged, and unobserved throughout its execution lifetime.

A second property matters just as much. Blunt enforcement that answers every anomaly by stopping the workload converts security events into availability events. Churchill's response model distinguishes the two: an attack that is *refused* costs the workload nothing, and an attack that has demonstrably *succeeded* is the one case where continuing to run is the greater risk. There is no observe-only mode for fatal events. Either the workload runs in a verified state, or it does not run at all.

---

2. Architecture

Churchill composes four cooperating layers across four trust domains.



The in-process sentinel is loaded into the protected application's address space. It runs continuous guards over code integrity, memory layout, timing, and privilege state, with sub-second detection latency and a self-termination path that cannot be cleanly intercepted.

The host sentinel is a privileged daemon that watches the application from outside its address space. It delivers the verdict on every execution attempt in the protected runtime before a binary's first instruction runs, monitors process lifecycle and mounts, attests the application's memory from outside, runs behavioral detectors, owns the termination engine, and maintains the local evidence chain.

The kernel enforcement layer uses BPF LSM, a facility of modern Linux kernels, to convert an entire class of detections into synchronous denials at the syscall boundary: the unauthorized action fails with a kernel error, the workload continues, and no reaction window exists at all.

The control plane is the off-host root of coordination: it holds enrollment state, locks per-host integrity baselines, stores the remote half of the evidence chain, runs statistically rigorous liveness detection over the fleet, and dispatches operator decisions (kill, unlock, lockdown) over authenticated channels. The dashboard provides fleet-wide visibility, evidence timelines, session

replay, and change-governance workflows.

The two sentinels monitor each other as well as the application. Freezing, pausing, or killing either one is detected by the other and treated as a tamper event. An attacker cannot disable protection without ending the workload, and cannot end the workload without leaving evidence.

2.1 Design Principles

Principle	Meaning
Deny first, terminate on compromise	Unauthorized actions are refused while the workload keeps running. Verified compromise of the workload itself triggers immediate, irreversible termination. There is no observe-only mode for fatal events.
Dual-control change	Nothing deploys without M-of-N approval signatures over the exact manifest. Approval keys stay with the quorum members; neither the host nor the control plane can forge an approval.
Mutual liveness	Both sentinels watch each other. Silencing either is itself a detected event.
Defense in depth	Independent layers (in-process guards, external attestation, kernel denial, remote baseline locking, operator dispatch) with no single point whose defeat is silent.
Cryptographic authentication everywhere	Every inter-component message is signed or sealed under keys bound to an authenticated session.
Forensic preservation	A hash-chained, dual-custody evidence record survives even the destruction of the workload.
Minimal persistent footprint	One on-host client binary and a per-host identity key on disk. Everything else arrives signed and lives in sealed memory.

---

3. Onboarding and Change Governance

Churchill's onboarding model is built on a principle stated plainly: discovery and configuration run on the host, and the host always connects outbound. Churchill never reaches into your infrastructure.

3.1 Registration

An operator issues a **provisioning code** from the dashboard: single-use, short-lived, and scoped to a tenant, environment, and source network. The code is a pairing credential only. On the host, the on-host client generates a persistent identity keypair, connects outbound to the control plane, redeems the code, and binds its public key to the enrollment record. No bundle, key, or secret ever travels to the host at registration; a stolen code buys an attacker nothing except a registration attempt the operator will see and the approval quorum will refuse.

The client then discovers the application in place (service definitions, binaries, configuration and data paths, and cryptographic hashes of all of them) and uploads the discovery record over the encrypted session. Configuration and hashes travel up; nothing secret travels down.

3.2 The Approval Quorum

A registered application deploys nothing by itself. The proposed protection manifest goes to a client-assigned change advisory board whose members each hold their own signing keys, on their own machines, outside Churchill entirely. Approval requires M-of-N signatures over the exact manifest hash; any member can veto, and unactioned requests expire. The control plane then countersigns the approved manifest with the signing key the host enforces against. The member receipts prove human authorization; the countersignature enforces it; and the manifest hash the quorum signed, the control plane signed, and the host verifies are the same bytes.

The result is a change-control property that holds even against privileged insiders: no administrator, no root user on the host, and no operator of the control plane can unilaterally alter what a protected host will run.

3.3 Deployment and the Sealed Runtime

After approval and publication, the host retrieves its deployment bundle over a mutually authenticated, encrypted session, using its own enrolled identity rather than any bearer credential. The bundle is a signed unit containing the sentinels, the protection manifest, and the application's runtime material. Verification checks both the signature and the signer identity against pinned public keys, rejecting bundles signed by any other party, even one holding a valid key for a different deployment.

Verified content is materialized into a **sealed, in-memory runtime**: kernel-enforced immutable memory that rejects modification from any process, regardless of privilege, for the lifetime of the system. The application's executables, libraries, and runtime material exist only in this sealed memory, never on disk, and are exposed through a private, read-only view that is itself protected against mount shadowing. A deployment is reported as deployed only after the host has verified, adopted, and acknowledged it.

The consequence for a root-level attacker who lands on the host afterward: the runtime cannot be modified in place, cannot be overlaid by a mount, and cannot be replaced on disk, because it is not on disk.

---

4. Runtime Protection Layers

4.1 In-Process Guards

The in-process sentinel runs four guard families concurrently, each independent so that defeating one does not defeat the rest.

Spatial integrity. The application's code and read-only data are cryptographically baselined once the process reaches steady state and re-verified continuously. Software breakpoints, patched instructions, injected shellcode, hooked library calls, unauthorized libraries, and executable memory appearing after initialization all produce mismatches, and mismatches are fatal.

Temporal integrity. Calibrated timing measurements detect the overhead that single-stepping and

instruction-level tracing cannot avoid imposing. The statistics are engineered to stay quiet under ordinary virtualization jitter while catching persistent anomalies, and implausible drift in the timing environment itself is treated as manipulation.

Privilege integrity. Debugger presence is both passively detected and actively blocked through multiple independent methods. The process's privilege and isolation state are baselined at startup; privilege expansion or a change in containment, the signatures of escalation and container escape, is fatal.

Artifact integrity. Pinned scripts, configuration, and data files are verified immediately before the application consumes them, collapsing the window between check and use, with per-artifact policy for warn versus terminate.

When an in-process guard fires, the sentinel alerts the host sentinel *before* acting, so that external termination proceeds even if the in-process kill path has itself been sabotaged. The termination sequence suppresses core dumps, wipes sensitive memory, and cannot be caught or handled by the (potentially compromised) process.

4.2 Host-Level Enforcement

Pre-execution gating. No binary executes inside the protected runtime without the host sentinel's verdict. Content is verified against the signed manifest in a way that closes the classic race between the check and the execution, and an unauthorized binary is denied before its first instruction: the attempt fails with an ordinary permission error, the workload continues serving, and the event enters the evidence chain.

Mount and lifecycle monitoring. Bind-mount shadowing defeats file-integrity checking that trusts paths, so the host sentinel baselines the mount table and treats any new mount appearing under a protected path as a tamper event; with the kernel layer present, the offending mount is denied outright. The protected process tree is tracked in real time: unexpected exec, unexpected exit, and credential changes are enforcement events.

External memory attestation. On a sub-second cadence, the host sentinel verifies the application's code and read-only memory from outside the process, comparing against reference values held beyond the application's reach. An attacker who can write application memory cannot also rewrite the baseline it is checked against, because that baseline is unreachable from the compromised address space. Applications can additionally register their own high-value memory regions for the same externally held verification. Attestation digests are also streamed to the control plane, which locks the first observed baseline and treats subsequent drift as an enforcement event rather than a new normal.

Behavioral detection. Two last-mile detectors complement the integrity guards: one tuned to the statistical signature of ransomware-encrypted output, and one tuned to scanning and exfiltration connection patterns. These detectors are secondary by design; integrity enforcement is the primary line. What they add is visibility into attack *effects* that tampering-technique coverage might miss.

4.3 Kernel-Boundary Denial

Where the kernel supports BPF LSM (mainline 5.7+ with the facility enabled; on by default in the RHEL 9 family), Churchill attaches an enforcement layer that turns detections into synchronous denials with microsecond latency. In this mode, the following are refused at the kernel boundary

before they take effect:

- debugger attachment and process-memory writes against protected tasks;
- lethal signals aimed at protected tasks from unauthorized senders;
- execution of any binary outside the signed manifest's allowlist;
- creation of writable-and-executable memory in the protected scope;
- mount operations and file modifications against the sealed runtime;
- high-risk privilege acquisition inside the protected scope;
- tampering with the enforcement surface itself from inside the protected scope;
- network egress to destinations outside the application's approved list.

The **egress allowlist** deserves emphasis because it closes the gap execution control alone cannot: living-off-the-land exfiltration uses binaries that are already authorized. With egress enforcement active, the workload can only talk to the destinations it was approved to talk to. Egress enforcement is deliberately deny-only: a refused connection is refused and recorded, never escalated to termination, because a blocked destination is evidence of an attempt, not proof that the workload itself is compromised. It is also introduced in a way that cannot sever a production workload from its legitimate traffic during rollout.

Churchill's kernel layer composes with the platform's existing security modules rather than replacing them: a denial by SELinux, AppArmor, or kernel lockdown is never converted into an allow.

Degradation is graceful and explicit. On kernels without BPF LSM, kernel-boundary denials degrade to detection and response by the sentinels, with latency bounded by monitoring cadence rather than syscall synchrony. The pre-execution gate, notably, remains a *denial* even without BPF LSM. Churchill installs and protects at reduced strength on older kernels rather than refusing to run.

---

5. The Response Model

Churchill's responses are graduated, and the graduation is the point.

Denial. Unauthorized executions, debugger attachments, memory-permission changes, mounts, and network destinations are refused at the boundary where they occur. The workload's availability is untouched. Every denial becomes an evidence event with full process context and computed hashes.

Termination and lockdown. When the running workload itself is verified compromised (code hash drift, attestation mismatch, a dead or silenced sentinel, credential or containment changes), the host sentinel executes an atomic termination: the protected scope is frozen so nothing can fork, exec, or dodge; every process in it is killed; the compromise and its reason are reported to the control plane; and the host enters **persistent lockdown**, refusing to bring the application back up across restarts and reboots until an authorized operator investigates and clears it with an audited, cryptographically signed unlock. The runtime's in-memory material can be wiped as part of the sequence. For a zero-tolerance workload, stopping *after verified compromise* is not an availability failure; it is the correct behavior, chosen deliberately and executed in milliseconds instead of at meeting-speed.

Fleet response. The control plane can dispatch signed kill and lockdown commands independently of local detection, whether on telemetry anomalies, on statistical liveness failure, or by operator decision. A compromise that persists across a clean redeploy escalates automatically to lockdown, on the reasoning that a host which comes back tampered has a deeper problem than its workload.

---

6. Evidence, Recording, and Observability

Hash-chained evidence. Every enforcement decision (allow, deny, kill) is recorded with full process context, expected-versus-actual hashes, and the SHA-256 of the preceding record. Modifying any record breaks the chain at that point with single-record granularity. The chain is held in **dual custody**: append-only on the host and mirrored to the control plane's store, so neither side can silently rewrite history. Critical events are flushed to the control plane immediately.

The word "tamper-evident" is used advisedly: Churchill does not claim magically unmodifiable storage. It claims, and delivers, that modification is *detectable*, cryptographically, at fine granularity, in two places at once. That is the property auditors can actually verify.

Interactive session recording. When a human touches a protected host, the actions themselves become evidence. Interactive shell sessions are captured live as timestamped terminal recordings, sealed and hashed under a separate privilege domain from the session being recorded, linked into the same evidence chain by hash, and streamed to the control plane over the authenticated channel with crash-safe resume. Input typed while the terminal has echo disabled (the normal password path) is masked before it is ever written, so secrets do not enter the record. Recordings are replayable from the dashboard, both live while the actor is still active and forensically afterward.

Dashboard. Fleet-wide integrity posture, per-host telemetry and liveness scoring, the evidence timeline with chain verification, session replay, change-governance workflows, and audited kill/unlock controls are exposed through a real-time dashboard backed by the control plane.

---

7. Cryptographic Posture

Churchill uses standard, widely reviewed cryptographic primitives throughout: the Ed25519 and X25519 curves, the SHA-256 family, modern authenticated encryption, and TLS 1.3. There is no proprietary cryptography. What matters architecturally is where keys live and what each signature proves:

Property	**How it is held**
Bundle and manifest authenticity	Signatures verified against pinned signer identities; a valid signature from the wrong signer is rejected
Change authorization	M-of-N receipts over the canonical manifest hash, signed with keys held by quorum members outside Churchill, countersigned for enforcement
Host identity	A per-host keypair generated on the host at registration; the private half never leaves the host; no shared credentials exist that would let

Property	How it is held
	one host impersonate another
Session security	Mutually authenticated, end-to-end encrypted channels with fresh ephemeral keys per session and replay rejection
Evidence integrity	SHA-256 hash chaining, held in dual custody across host and control plane
Signing-key custody	Pluggable backends: encrypted file, TPM 2.0, or PKCS#11 HSM

Fresh ephemeral keys per session mean compromise of a session key exposes nothing about past or future sessions. Key rotation is a release or change-control event by design; there is no runtime configuration path an attacker with file access could use to substitute trust anchors.

---

8. Deployment and Requirements

- Linux kernel **5.0+**, cgroup v2, systemd 240+.
- **Full-strength kernel enforcement** requires kernel 5.7+ with BPF LSM built and active. This is the default on the RHEL 9 family (RHEL, Rocky, Alma); on Ubuntu and Debian it is a boot-parameter change, not a rebuild. Hosts without it run with userspace enforcement at documented reduced strength; they are supportable, not unsupportable.
- Architectures: **x86_64 and IBM Z / LinuxONE (s390x)** today, built and validated dual-architecture as a matter of engineering process, with every integration test running on both.
- Applications are protected **without source modification**: the application is wrapped under the in-process sentinel at launch, and all of its binaries, libraries, scripts, and data files are hash-pinned in the signed manifest.
- The protected application's service is bound to the host sentinel at the service-manager level, so the application cannot run unwatched, and a sentinel failure fails closed rather than open.

Performance Characteristics

Churchill is engineered against a strict overhead budget:

- Kernel-layer checks add sub-microsecond latency per intercepted operation; measured steady-state overhead under heavy I/O is below 3% on a 4-vCPU x86_64 host.
- Execution gating adds single-digit milliseconds to launching protected binaries only; ordinary I/O is untouched.
- Continuous monitoring (in-process guards and external attestation) runs within a small fixed CPU budget, sized for single-vCPU hosts.
- Control-plane heartbeats are a few hundred bytes per interval per host.

---

9. Threat Model and Boundaries

Churchill defends a workload against adversaries with user-space access to the host, up to and including root, after deployment:

Threat	Response
Debugger attachment	Denied at the kernel where supported; detected by multiple independent methods otherwise; fatal
Code injection and hooking	Detected by continuous code, library, and memory-layout verification; unauthorized executable memory denied at the kernel
Unauthorized execution	Denied pre-execution; the workload keeps running
Binary or artifact substitution	Denied by manifest verification and the sealed runtime
Memory tampering	Denied at the kernel where supported; detected by external attestation on a sub-second cadence otherwise
Filesystem shadowing	Mount operations denied against the sealed runtime; baseline watchdog otherwise
Privilege escalation and container escape	Privilege and isolation baselines; high-risk privilege acquisition denied at the kernel
Living-off-the-land exfiltration	Egress allowlist denial (deny-only, recorded); behavioral detectors
Sentinel tampering or silencing	Mutual liveness; shared fate; independent external and remote detection
Unauthorized change to the protection itself	M-of-N quorum over the exact manifest; member-held keys
Evidence tampering	Hash-chain verification failure, in dual custody

Explicit non-goals. Churchill does not defend against adversaries who start with kernel-level control (malicious kernel modules, `/dev/mem`, kernel-text manipulation), physical access (cold boot, DMA, JTAG), or hardware and firmware implants. The kernel layer raises the bar against ring-0 tampering but is not a substitute for kernel lockdown. Churchill consumes hardware identity where present but does not itself measure the boot chain: organizations requiring measured boot should layer Secure Boot, kernel lockdown, and TPM-based attestation alongside it. Perimeter network security and data-loss prevention remain the operator's responsibility; Churchill's egress control is a workload-scoped allowlist, not a network IDS.

These boundaries are stated because they are load-bearing: a product that claims to stop everything can be trusted about nothing. Within its stated model of protecting a high-assurance application against post-deployment tampering by user-space adversaries, Churchill provides continuous, cryptographically verifiable runtime integrity that no single mechanism can deliver alone.

---

10. Conclusion

Churchill treats runtime integrity as a property to be *enforced continuously*, not a log line to be reviewed later. Unauthorized code does not run; unauthorized destinations are unreachable; unauthorized changes never deploy; and every attempt at any of the three is refused while the workload keeps serving, then preserved in an evidence chain that two independent parties would have to collude to falsify. The one event that stops a protected workload is the one event where stopping is the only defensible choice: verified compromise of the workload itself. Even then, the decision is executed atomically, recorded durably, and reversible only by an accountable human.

Either the workload runs in a verified state, or it does not run at all. Everything else keeps running.

Churchill is built entirely on standard, in-tree Linux kernel security facilities and standard, widely reviewed cryptographic primitives: no custom kernel modules, no proprietary cryptography. It requires Linux kernel 5.0+, cgroup v2, and systemd 240+; kernel-boundary enforcement requires kernel 5.7+ with BPF LSM active.

wgds.tech