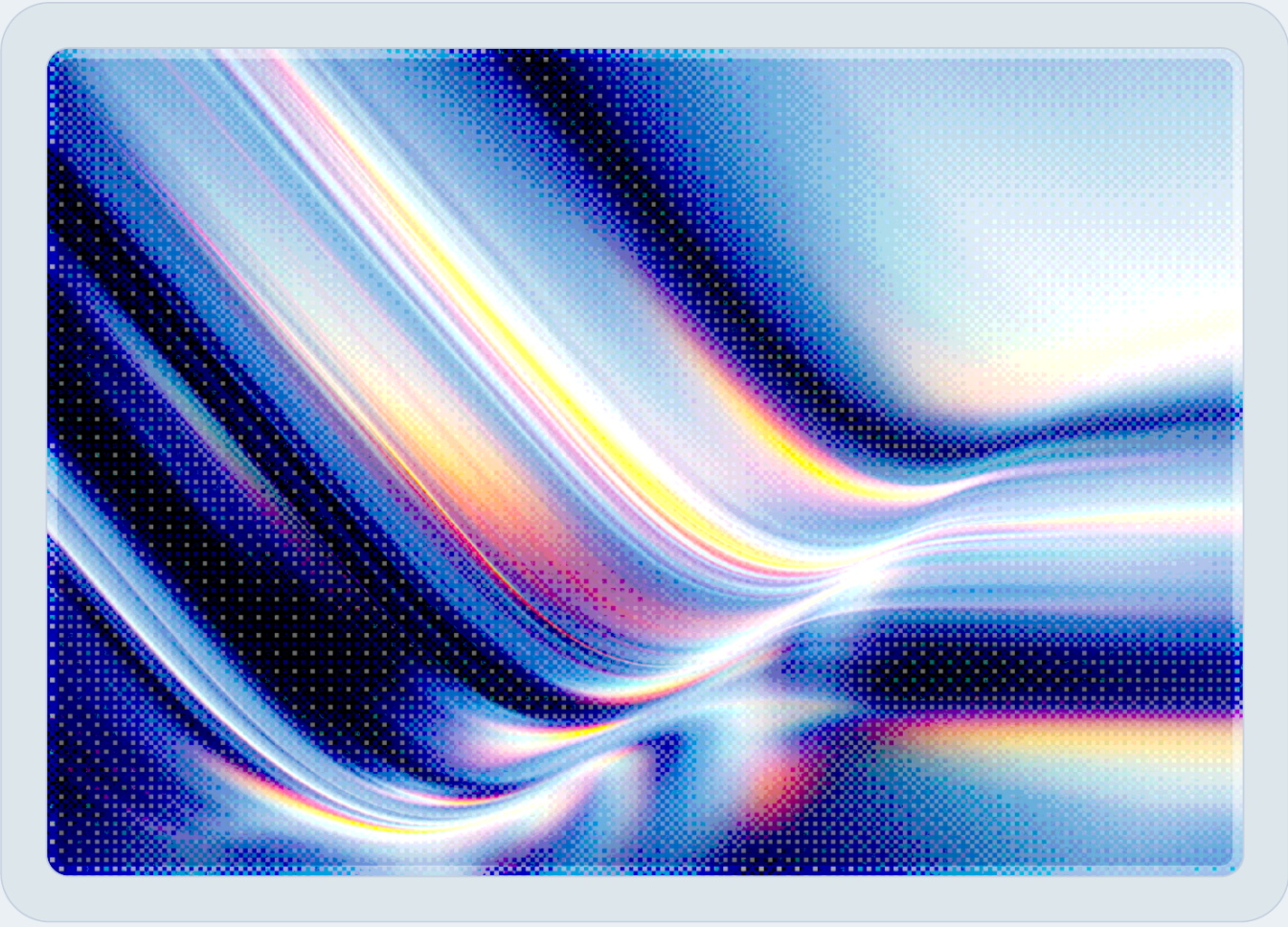
CSAI Foundation | Cloud Security Alliance

Agent-Written Kernel Shields: Correlated-Failure and Trust Risks

Security implications of self-defending infrastructure after the Shirudo launch

2026-11-24

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

© 2026 Cloud Security Alliance. Some rights reserved.

You may download, store, display, view, print, redistribute, and link to this document in its original, unmodified form, provided that attribution to the Cloud Security Alliance is maintained and all trademark and copyright notices remain intact.

This document may not be modified or altered. You may quote portions of the document as permitted by the Fair Use provisions of the United States Copyright Act, provided that attribution is given to the Cloud Security Alliance.

This document may be shared on professional and social media platforms in its original form with attribution.

This document was generated with AI assistance and has not undergone official CSA review and approval processes.

Key Takeaways

On November 23, 2026, Cisco and OpenAI released Shirudo, an open-source, eBPF-based runtime in which AI agents can find a vulnerability, write a temporary "shield" that blocks the behavior an exploit depends on, test it, and deploy it to running Linux systems without a reboot [1][2][3]. The project addresses a real problem. Recent results suggest that AI-driven exploit generation can produce working kernel privilege escalations far faster than upstream fixes land. In one October 2026 run, a single pipeline produced exploits for 948 vulnerabilities, of which 61 had been patched ten days later [4][20]. The interval before a fix is available and deployed is a period of acute exposure [4][5]. Shirudo's design includes post-quantum code signing, a kernel-enforced signed loader, and TPM-backed measurement, and its threat model documents its limits explicitly [6].

Even so, agent-written kernel shields move the location of systemic risk rather than removing it. Three concerns stand out for security leaders. First, a single shield that is signed once and rolled out "to all affected nodes" [3] creates the same correlated-failure pattern that turned one faulty content update into the July 2024 outage of an estimated 8.5 million Windows devices [7][8]. Second, the project itself states that it "establishes provenance, not correctness" [6]. A valid signature proves who approved a shield; it does not prove that the agent reasoned correctly about what the shield blocks. Third, the signing key, the generating model, and any shared catalog of shields each become a high-value point of concentration. The project's own documentation warns that "any artifact the key ever signed loads" [6].

CSA's assessment is that agent-written runtime mitigations are likely to become a standard part of vulnerability response, given the patch-gap pressure described below and parallel upstream efforts such as the kernel killswitch proposal [10]. Organizations should prepare for them. They should be governed as privileged production changes to the kernel, with staged rollout, independent validation, enforced expiry, and segmented signing authority. They should not be treated as self-validating security content. Shirudo describes itself as pre-release [6], so it is too early for production adoption decisions. It is the right time to set governance policy.

Background

The Patch-Gap Problem Shirudo Targets

The pressure behind self-defending infrastructure is measurable. CSA's research note *948 Paths to Root* summarized reporting on an OpenAI pipeline that, in a single October 2026 run, produced 1,169 working local privilege escalation exploits covering 948 distinct vulnerabilities in a current Linux kernel, figures that OpenAI's November 2026 disclosure also put at the end of a funnel of 2,308 candidate vulnerabilities [4][20]. Ten days later, 61 of those bugs had been patched upstream [4]. Greg Brockman's August 2026 essay "The Defender's Window" made a similar argument. It said broadly available models were "only a few months behind the frontier" in offensive capability, and that defenders should adopt agents now [9]. In the Shirudo announcement, Brockman stated the operational problem directly: "finding a bug today doesn't mean you can patch every system tomorrow" [1].

The Linux community has been looking at the same gap from another direction. In May 2026, stable-kernel co-maintainer Sasha Levin proposed a "killswitch" patch that would let administrators disable specific vulnerable kernel functions at runtime until a fix is installed [10]. The discussion is useful context for Shirudo. Commenters raised concerns about giving root more power to alter a running kernel, about attackers who already hold root misusing the mechanism, and about collateral damage. One example raised by commenters was that nf_tables, which the patch described as rarely used, underlies iptables on modern Debian systems [10]. Hackaday reported that the patch was at least partly LLM-generated, citing an earlier report [11]. Upstream reviewers have therefore already debated runtime kernel mitigation, its risk of collateral damage, and AI-authored kernel changes, independently of Shirudo.

What Shirudo Is and How It Works

Shirudo is described as "a rewrite of Tetragon designed for agents," building on the eBPF runtime-security project that Isovalent open-sourced in 2022 [3]. In the workflow Cisco and OpenAI describe, a control plane driven by OpenAI's Codex ingests vulnerability disclosures, builds and validates an exploit reproducer, chooses a mitigation point (a kernel eBPF hook or, for Java workloads, runtime bytecode injection), and repeats mitigation development "until a successful mitigation is found." The resulting shield is then signed and rolled out while application health is monitored [2][3]. The launch demonstration involved a Java vulnerability the vendors call "Java4Shell." Codex generated a guard on Java's Base64 decoder and attached it to a running Elasticsearch process. Help Net Security noted that "the people who built the tool also ran that test and set its conditions" [5].

The repository documentation (where shields are called "upatches") describes a security architecture that is more conservative than the launch announcements suggest [1][2][6][12]. Signing uses ML-DSA-87 (FIPS 204) only. The private key stays on the operator host and never reaches a target. The kernel restricts BPF program loads to signed loaders checked against a dedicated `.bpf` keyring, and a TPM 2.0 records loads and removals so that a remote party can verify them through attestation. Packages are pulled from OCI registries that are treated as untrusted. Production use requires a TPM 2.0 chip and a kernel carrying Shirudo's own patches [5][6], including LSM hooks that the documentation says upstream Linux does not have [6]. The project is licensed BSD-2-Clause with BPF programs dual-licensed BSD/GPL, and its README states that it "is pre-release" [2][6].

The public accounts differ on how much autonomy the system has. OpenAI's announcement says shields deploy "once an operator approves" them, and Help Net Security repeats that point [2][5]. The Isovalent engineering blog says the process "can be designed to be more or less autonomous" and shows a fully automated run [3]. Cisco's executive blog frames the broader industry shift by saying agents "are the operators now" [12], though it does not specify Shirudo's approval workflow. These differences matter, because the governance controls recommended below depend on where human authority actually sits in a given deployment.

Security Analysis

Correlated Failure: One Artifact, Many Kernels

The central systemic concern is that shields are designed to roll out to every affected node at once, which for a common kernel component may be most of a fleet. Isovalent describes the mitigating shield being "signed and securely rolled out to all affected nodes" [3]. OpenAI states a goal "to make tested shields available alongside vulnerability disclosures" so that "others don't have to start from scratch" [2]. Each of these choices serves the goal of closing the patch gap quickly. Together, they mean the same kernel-resident logic could run at once across many hosts, many tenants, and eventually many organizations, and it would be pushed out during the urgent hours after a disclosure, when there is least time to validate it.

The CrowdStrike Channel File 291 incident shows how this goes wrong. CrowdStrike's root cause analysis found that the new template type defined 21 input fields, while the sensor code that invoked it supplied only 20 values, and that a logic error in the content validator allowed the problematic template instance through. When the content interpreter tried to read the missing 21st value, the resulting out-of-bounds read crashed hosts worldwide [7]. Microsoft estimated that 8.5 million devices were affected, under one percent of Windows machines [8]. CrowdStrike's remediation included staged deployment of rapid-response content, which it had not previously applied to that class of update [7]. Agent-written shields

serve the same purpose as rapid-response content: fast, broad protection after a disclosure. Their safety depends on the validation pipeline being correct and on rollout being staged. In 2024, the validator failed and staged rollout was absent [7].

eBPF does reduce some of this risk, but it does not remove it. The eBPF verifier checks memory safety and termination, which makes outright kernel crashes from shield code less likely than with a kernel module. The verifier is still software and has had serious bugs. CVE-2023-2163, for example, was an incorrect pruning flaw that allowed unsafe code paths to be "incorrectly marked as safe," leading to arbitrary kernel memory access [13]. eBPF-based security products have also been involved in fleet instability before. In mid-2024, Red Hat customers saw kernel panics on RHEL 9.4 associated with the Falcon sensor's BPF-based user-mode component, which The Register attributed to a kernel BPF bug [14]. More fundamentally, the verifier cannot judge intent. A shield that passes verification can still block a code path that legitimate workloads depend on, as the nf_tables example from the killswitch debate shows [10]. Shields delivered as Java bytecode injection do not go through the kernel verifier at all; the documentation says they are verified in user space only [6].

Correlation can also occur when no artifact is shared. If many organizations ask the same frontier model to write a shield for the same disclosed vulnerability, the model's blind spots may repeat across their outputs: the same mis-chosen hook, the same overlooked legitimate caller, or the same bypass through an alternative code path. CSA has not seen empirical data on the diversity of model-generated mitigations, so this is an inference rather than an observed failure. It follows the same logic as CSA's earlier analysis of concentration in AI-driven patching [15]. Relying on one model as the main source of defensive logic would create a form of monoculture.

Trust: Provenance Is Not Correctness

Shirudo's signing design addresses the threat it names, which is an attacker with root on a target who does not hold the signing key [6]. It does not address, and does not claim to address, whether the signed content is correct. The project's threat model says so plainly ("shirudo establishes provenance, not correctness"), and also says that host telemetry is only "the target's own account" except for the parts the TPM signs [6]. This means the meaning of an operator's approval changes. In a human-written workflow, the signer has usually read the change. In an agent-written workflow running at machine speed, the signer may be approving evidence that the same agentic pipeline generated: the reproducer, the shadow-mode results, and the health metrics. When the system that writes the shield also grades it, reviewers can mistake internal consistency for independent assurance.

The inputs to that pipeline also matter. Cisco describes agents that "scan open-source code and take in vendor disclosures" [12]. Those are inputs an adversary can influence. A crafted advisory, a poisoned reproducer, or prompt-injected content in a repository could push an agent toward a shield that looks

effective in testing but leaves an alternative exploit path open. A worse outcome would be a shield that disables a defensive capability, such as a logging or audit path, under the label of mitigation. CSA is not aware of such an attack against Shirudo; the risk follows from the architecture. The vendors' public invitation to "show us where it breaks legitimate traffic" [1] indicates that real-world behavior is still being characterized, and CSA has seen no published evaluation of the pipeline's resistance to adversarial inputs.

Key Concentration and Lifecycle Gaps

The signing key becomes a fleet-level master key for the kernel. The repository's threat model notes that a compromised key affects "every target whose `.bpf` keyring holds the key" [6]. As CSA reads the published security documentation, the current design has no per-artifact expiry, version floor, or per-target authorization. Retiring a shield effectively means revoking the key that signed it, and anti-rollback protection for signed configuration is deferred. The documentation also describes boot-time behavior in which a failure to initialize lets the target continue booting without enforcement [6]. The project documents these limitations explicitly, and they may change as the pre-release software matures. They also conflict with Cisco's statement that a shield "retires when the real patch lands" [12], which, based on the current repository, would have to be enforced by operator tooling rather than by the runtime. Fleet rollout order, scheduling, and cross-host consistency are likewise outside the project's scope [6], so the controls that matter most for correlated failure are left to each adopter.

The table below summarizes the main risks, how each arises, and what is publicly known.

Risk	Mechanism	Evidence / Status	Primary Control
Fleet-wide functional outage	One shield blocks a legitimate path on all affected nodes at once	Fleet rollout described by vendor [3]; CrowdStrike precedent [7]	Ring-based staged rollout with automated halt
Verifier or loader defect	Bug in eBPF verifier or signed-loader path admits unsafe code	Historic verifier CVEs [13]; required out-of-tree kernel patches [6]	Kernel patch currency; limit shield program types
Incorrect but validly signed shield	Agent misjudges exploit or legitimate callers; self-graded evidence	Project disclaims correctness [6]; vendor-run test only [5]	Independent validation; human review of scope
Adversarial steering of the	Poisoned disclosure, reproducer, or repository	Inferred from architecture [12]; no	Treat agent inputs as untrusted; diff-

Risk	Mechanism	Evidence / Status	Primary Control
agent	content	known incident	based review
Signing key compromise	Any artifact the key signed will load on every enrolled host	Documented in threat model [6]	HSM custody; key per fleet segment; dual control
Stale or rolled-back shields	No built-in expiry or anti-rollback	Documented as residual risk / deferred [6]	External expiry enforcement; attestation audits
Model monoculture	Many organizations use one model to write mitigations for the same bug	Inference; analogous to patch concentration [15]	Diverse validation; treat shared shields as untrusted

Recommendations

Immediate Actions

Organizations should first establish where they stand. Security teams should inventory every eBPF-based agent already running in production, including EDR sensors, observability tools, and network policy engines. They should record who can load programs, whether loads are signed, and how updates are staged. Shirudo is pre-release and needs patched kernels and TPM 2.0 [6], so CSA recommends limiting it to lab and research environments for now. Teams experimenting with it should do so on isolated, representative workloads, with a tested removal path. Before any pilot, the organization should write down a *shield authority policy* that answers three questions: who may approve a kernel-resident mitigation, what evidence that approval requires, and which hosts are eligible. Without such a policy, the autonomy setting will default to whatever the tooling is configured to do [3].

Short-Term Mitigations

Over the next one to two quarters, organizations should build the controls the runtime leaves to them. The most important is staged deployment. Shields should move through canary, early-ring, and broad-ring stages, with automated halts tied to application health signals. Emergency timelines should shorten

the time spent in each ring rather than skip rings, which is the lesson CrowdStrike drew from Channel File 291 [7]. Organizations should also separate generation from validation. A shield written by one model should be checked by a different method, such as a human reviewer, a second model from a different provider, or a reproducer and traffic-replay suite built independently of the generating agent, before it moves from monitoring to blocking.

Signing authority deserves the same care as a code-signing root. ML-DSA signing keys should be held in hardware security modules that support ML-DSA, or, where such support is not yet available, in the strongest available isolated signing environment, with dual control in either case. Separate keys should be issued per environment or fleet segment, so that one compromise cannot reach every host. Because the runtime does not yet enforce expiry [6], each shield should carry an explicit expiry date and a linked patch ticket in operator tooling. TPM attestation reports should be compared against that inventory on a schedule, so that stale, unexpected, or rolled-back shields are found. Teams should also check boot-time behavior directly and alert when a host comes up without enforcement.

Strategic Considerations

Over the longer term, organizations should treat shields published "alongside vulnerability disclosures" [2] as untrusted third-party code. They should accept them only after local validation, and should track their provenance much as they would track SBOM entries. Procurement and vendor-risk teams should ask any provider of agent-generated mitigations which model produced each shield, what independent validation it received, how rollout is staged, and how quickly a bad shield can be withdrawn.

Beyond their own environments, organizations should also support upstream and community work on the gaps described here. Useful areas include per-artifact expiry and anti-rollback in the runtime, standard formats for recording shield provenance, and kernel-community consensus on runtime mitigation interfaces, an area the killswitch debate shows is still unsettled [10]. Boards and risk committees should consider agent-written kernel mitigations as part of AI concentration risk alongside model-provider dependence, since a common model, a common catalog, and a common signing practice can each become a single point of failure.

CSA Resource Alignment

The most directly related CSA work is *The Clearinghouse Rush: Concentration Risk in AI Patching* [15], which examined how commercial vulnerability clearinghouses pool pre-disclosure data and concluded that actuation, meaning the speed from finding to fix, matters more than the size of the pool. Agent-written shields are the actuation layer that analysis anticipated, and they extend its concentration concern from

vulnerability *data* to kernel-resident *defensive code*. The clearinghouse note's guidance on evaluating dependency on a single intermediary applies directly to shared shield catalogs and to reliance on a single generating model.

AI Provider Concentration Risk: Enterprise Resilience [16] provides the resilience framework for the model-monoculture and provider-dependence issues raised above. Its architectural, contractual, and governance recommendations transfer to organizations whose emergency mitigation capability now depends on one AI provider's agent. *948 Paths to Root* [4] and *Linux DirtyFrag Wave: FIM Evasion on AI Infrastructure* [17] document the kernel privilege-escalation pressure that motivates runtime shielding. They also show why a root-capable adversary, the adversary Shirudo's signing model is designed to stop, is a realistic assumption for AI infrastructure.

For control mapping, CSA's **AI Controls Matrix (AICM) v1.1** [18] supplies the relevant vocabulary. Its Change Control and Configuration Management, Threat and Vulnerability Management, Cryptography and Key Management, and Supply Chain domains together cover staged shield rollout, emergency mitigation governance, signing-key custody, and acceptance of third-party shields. **MAESTRO** [19], CSA's agentic AI threat modeling framework, is the appropriate tool for modeling the generating agent itself. That includes its exposure to poisoned disclosures and reproducers, the integrity of its self-produced validation evidence, and the trust boundary between the agentic control plane and the kernel it changes.

References

[1] Cisco. "Cisco and OpenAI Launch Shirudo, A New Project to Safeguard Agentic AI." Cisco Newsroom, November 23, 2026.

[2] OpenAI. "OpenAI and Cisco introduce Shirudo." OpenAI, November 23, 2026.

[3] Graf, Thomas and Liz Rice. "Cisco and OpenAI introduce Shirudo." Isovalent (Cisco) Blog, November 23, 2026.

[4] Cloud Security Alliance. "948 Paths to Root: Industrialized AI Linux Exploit Generation." CSA AI Safety Initiative, November 2026.

[5] Zorz, Mirko. "Linux patches can take weeks, and OpenAI and Cisco built Shirudo to help in the meantime." Help Net Security, November 24, 2026.

[6] Shirudo Project. "shirudo: eBPF-based security platform (README, SECURITY.md, and Documentation/security threat and security models)." GitHub, accessed November 24, 2026.

[7] CrowdStrike. "Channel File 291 Incident Root Cause Analysis." CrowdStrike, August 6, 2024.

[8] Weston, David. "Helping our customers through the CrowdStrike outage." Microsoft, July 20, 2024.

[9] Brockman, Greg. "The Defender's Window." OpenAI, August 17, 2026.

[10] Corbet, Jonathan. "killswitch for short-term emergency vulnerability mitigation." LWN.net, May 8, 2026.

[11] Posch, Maya. "After Stumbling From CVE To CVE Will Linux Get A Kill Switch?" Hackaday, May 16, 2026.

[12] Patel, Jeetu. "Expanding the Defender's Window with Self-defending Infrastructure." Cisco Blogs, November 23, 2026.

[13] NIST. "CVE-2023-2163 Detail." National Vulnerability Database, September 20, 2023.

[14] Sharwood, Simon. "CrowdStrike's Falcon Sensor also linked to Linux kernel panics and crashes." The Register, July 22, 2024.

[15] Cloud Security Alliance. "The Clearinghouse Rush: Concentration Risk in AI Patching." CSA AI Safety Initiative, July 2026.

[16] Cloud Security Alliance. "AI Provider Concentration Risk: Enterprise Resilience." CSA AI Safety Initiative, June 2026.

[17] Cloud Security Alliance. "Linux DirtyFrag Wave: FIM Evasion on AI Infrastructure." CSA AI Safety Initiative, June 2026.

[18] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.1." Cloud Security Alliance, accessed November 24, 2026.

[19] Cloud Security Alliance. "Agentic AI Threat Modeling Framework: MAESTRO." Cloud Security Alliance, February 6, 2025.

[20] SecurityWeek. "AI Agents Got Root on Linux via 948 Kernel Flaws in OpenAI Research." SecurityWeek, November 2026.