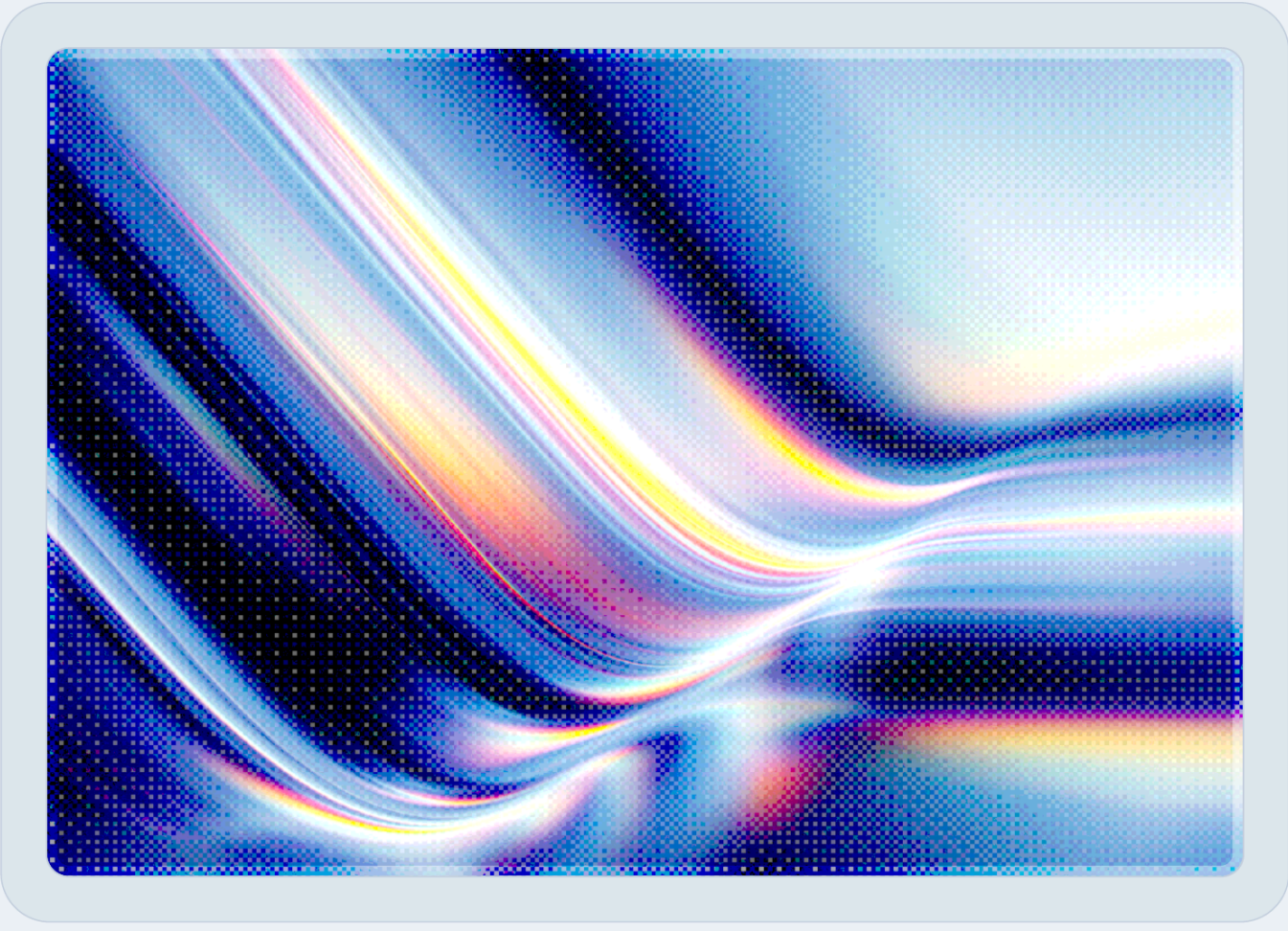
CSAI Foundation | Cloud Security Alliance

Hex.pm Registry Server Compromise: When the Registry Is the Target

Security Implications and Guidance After CVE-2026-129514

2026-11-25

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, an unidentified attacker exploited CVE-2026-129514 in the hexpm server, the application that runs Hex.pm, the package registry for the Elixir and Erlang ecosystems. The attacker ran operating system commands on six production web servers over about thirteen minutes. Hex.pm maintainers had patched every server by 18:31 UTC, roughly fifteen minutes after the first exploit request [1][3]. The flaw sat in a custom request-body parser that ran before routing and authentication. It accepted forged Elixir structs, which let an unauthenticated POST request reach a chain of library "gadgets" that ended in arbitrary code execution [1]. The Erlang Ecosystem Foundation rated the vulnerability CVSS 10.0 [2][4].

Hex.pm's forensic review found no unexplained database changes and no tampered packages or registry files. It also found no persistent shells and no new cloud credentials [1]. However, the maintainers cannot rule out that the attacker read email addresses, password hashes, API key hashes, two-factor authentication (2FA) secrets, sessions, and audit logs [1]. The most consequential open issue is the registry signing key. It was within the attacker's reach and has not yet been rotated, because Hex lacks a key-rotation mechanism. That mechanism is now being built [1].

The immediate actions differ by audience, and the table below summarizes them. Individual account holders face credential and phishing risk, while build teams, self-hosted operators, and application developers each face a distinct integrity or exposure question.

Audience	Recommended action
Users with Hex.pm accounts	Change your password and re-enroll your 2FA authenticator now. All sessions and client tokens have already been revoked [1].
Build and platform teams	Until patched Hex clients and a new signing system ship, fetch packages only from `repo.hex.pm` and treat third-party mirrors as untrusted, since mirror content is only as trustworthy as the registry signature that clients check [1].
Operators of self-hosted hexpm instances	Deploy a build that includes commit `379b0a2` [3][5].

Audience	Recommended action
Elixir application teams	Review exposure to the related gadget-chain fixes in `db_connection`, `ash`, and `flop` [1][3].
Registry operators in every ecosystem	Treat the registry server and its signing keys as primary supply chain targets in threat models.

Taken together, these actions address the confirmed exposure of account data and the unresolved question of signing-key integrity. The remainder of this note explains why the second issue deserves the most sustained attention.

Background

Hex.pm is the central package repository and API for Elixir, Erlang, and Gleam. Its client tooling is built into `mix`, `rebar3`, and the Gleam build tool. Like other registries, it has two separate faces. One is an HTTP API that handles accounts, authentication, and package publishing. The other is a read-only repository, served from a CDN at `repo.hex.pm`, which delivers registry metadata and package tarballs [8]. Clients do not verify the repository by trusting the host that serves it. Instead, they verify registry resources against an RSA public key shipped inside the Hex client. Hex's public-key page says that the active key was released on March 3, 2016, has never been changed, and that no key has ever been revoked [6]. Hex lists unofficial mirrors such as jsDelivr and UPYUN alongside its official Fastly CDN [8]. Because trust rests on the signature rather than the host, this design is presumably what allows such mirrors to be used without trusting their operators. It also means one long-lived private key anchors trust for the whole ecosystem.

The incident came seven months after a third-party security audit of Hex. The Erlang Ecosystem Foundation's Ægis initiative ran the audit, Alpha-Omega under the OpenSSF funded it, and Paraxial.io and zentrust partners GmbH carried it out [9][10]. That audit led to fixes for several server-side issues. They included an OAuth scope escalation (CVE-2026-21621), non-expiring password reset tokens (CVE-2026-21622), and an oversized-package denial of service (CVE-2026-23940). It also fixed a client-side unsafe deserialization bug in `hex_core` (CVE-2026-21619), which its advisory rated low severity because no released version was known to allow code execution [10]. The flaw behind CVE-2026-129514 does not appear among the audit's published findings. The audit reviewed the Hex registry, clients,

documentation infrastructure, and supporting systems [9], but it is not clear from public reports whether the request-body parser was in scope. That flaw, a gap in the safety checks applied to externally supplied Erlang terms, had been present since a check was removed on February 23, 2016 [1].

According to Hex.pm's postmortem, the attacker's two source addresses had repeatedly downloaded hexpm's own dependencies since at least November 19, the start of the log review window. The maintainers interpret this as the attacker building a local copy of the open-source server to develop and test the exploit offline [1]. The first exploit request arrived on November 23 at 18:16:00 UTC. Evidence of command execution appeared at 18:16:46, and the attack totaled 69 malicious requests through 18:29:33 [1]. A restart during the response briefly brought up three new servers still running the vulnerable build. This is why six servers were affected rather than three [1]. At 19:07 UTC, Hex.pm received an emailed security report describing the same vulnerability. The postmortem does not say who sent it or whether it is connected to the attacker [1][4]. Hex.pm notified Sweden's data protection authority (IMY), CISA, and ENISA the same evening and published the CVE on November 24 [1].

Security Analysis

An Unauthenticated Path Through "Safe" Deserialization

The root cause is a reminder that a "safe" decoding mode protects only against the threats its designers had in mind. Hexpm accepted request bodies in the `application/vnd.hex+erlang` content type, which is Erlang's External Term Format. It decoded them with `Plug.Crypto.non_executable_binary_to_term/2` in `:safe` mode [1]. That decoder rejects functions and new atoms, which blocks the best-known Erlang deserialization attacks. The postmortem notes, however, that it never checked maps. An Elixir struct is just a map whose `__struct__` key names a module [1]. An attacker could therefore send a term that the application would treat as a struct of any loaded type, with whatever field values the attacker chose. Because the parser ran before routing and authentication, any route accepted the payload from an anonymous client [1].

Turning forged structs into code execution needed a gadget chain spread across several widely used libraries. The postmortem gives the sequence step by step [1]. `Plug.MethodOverride` read a `_method` field from the forged body, which raised an `UndefinedFunctionError`. Error reporting through Sentry, Bandit's logger, and the JSON log formatter then called `inspect` on the malicious data. This dispatched to the `Inspect` implementation for `Ecto.Query.DynamicExpr`, which enumerated a field without first checking that it was a list. A

forged `DBConnection.Stream` placed in that field led into `DBConnection` internals. There, an attacker-supplied `{module, function, args}` logging callback ran even on the error path, which gave command execution as the BEAM process user [1]. Each malicious request still returned an ordinary HTTP 500 to the attacker [2][4]. The structure resembles well-known Java and .NET deserialization gadget chains. The chain needed several libraries working together, and no single step would have given code execution on its own. Several of the links have since received their own CVEs, which shows how gadget-chain risk spreads across error-handling and observability code that does not expect attacker-controlled structs.

Why Registry Infrastructure Changes the Risk Calculus

Many of the 2026 supply chain incidents CSA has covered, such as those described in *Miasma* and *Single Points of Failure*, involved stolen maintainer credentials or malicious packages, which the registry then distributed faithfully [11][13]. Registry-operator weaknesses have appeared before, for example in the RubyGems.org CDN key exposure [14]. The Hex.pm compromise is nonetheless distinct, because the attacker gained code execution on the registry service itself, the service that every consumer of the ecosystem trusts. The table below summarizes what that position could and could not provide, based on Hex.pm's disclosed findings.

Asset within the attacker's reach	Hex.pm finding	Residual risk to consumers
Package tarballs and registry files	All 62 package files and 186 registry files written since 18:00 UTC matched the database. No older release changed [1]	Low for existing releases. Checksums in `mix.lock` add protection for locked versions [7]
Database contents (users, hashes, 2FA secrets, sessions)	No unexplained row changes. Reads cannot be ruled out [1]	Phishing, password cracking, and 2FA bypass against maintainers. Sessions already revoked [1]
API keys	Stored as HMAC-SHA256 hashes of 128-bit random values. Hex.pm says rotation is not needed [1]	Low
Infrastructure credentials (AWS, database, storage)	Rotated between November 23 and 24 [1]	Low if rotation was complete. Direct cloud-storage reads are not logged [1]

Asset within the attacker's reach	Hex.pm finding	Residual risk to consumers
Private packages and documentation	Reads from compromised servers would be indistinguishable from normal access [1]	Possible confidentiality loss. Hex.pm is contacting affected organizations [1]
Registry signing key	Within reach and not yet rotated [1][3]	If the key was exfiltrated, unmodified clients could accept forged registry metadata delivered through an attacker-controlled path

The signing key row needs the most attention. Hex.pm has not said that the key was stolen, and the forensic evidence shows no misuse. The postmortem says plainly that environment-variable reads and shell commands were not logged [1]. The Erlang Ecosystem Foundation advisory states that the compromised process could read the application environment, which held the database URL, the storage credentials, and the registry signing key [2][4]. If an adversary holds that key, they could in principle present forged registry metadata to any client that reaches the registry through a path they control. Such a path could be a compromised or malicious mirror, an internal proxy that caches from untrusted sources, or a network position that can also defeat TLS, for example through a TLS-intercepting corporate proxy or a compromised CDN or certificate. Hex's checksums protect a version only after a project has locked it. The FAQ describes them as a way to detect a changed version of a package the user has already fetched and locked [7]. New dependency resolution, fresh CI environments without committed lockfiles, and upgrades are where forged metadata would matter most. Fetching directly from `repo.hex.pm` over HTTPS limits this exposure, which is why Hex.pm's interim advice tells users to fetch only from that host until new clients that support key rotation are released [1].

The incident also shows a structural weakness that is not unique to Hex. A signing key that has not changed in more than a decade, and that cannot be rotated without a coordinated client release, means that any server-side compromise able to reach the key becomes an ecosystem-wide trust problem. That remains true even when no package is modified. The fact that the key was readable from the web application's environment shows that signing material and the internet-facing application shared one trust boundary [2]. Offline or HSM-backed signing would separate them.

Observability as Both Detection Channel and Attack Surface

Two features of this incident point in opposite directions. First, the error-tracking pipeline was part of the gadget chain. `inspect` calls made for logging and Sentry reporting are what reached the vulnerable `Inspect` implementation [1]. Second, the same Sentry instance logged every exploit request and enabled fast detection. The first error was recorded at 18:16:00, and the team's first response on the servers came at 18:23:55 [1]. The forensic precision of the postmortem also came from good preparation: a pre-incident database snapshot, file creation times matched to request timestamps, and full comparison of package and registry files [1]. That preparation is what allowed Hex.pm to say with confidence that no published artifact changed. The same evidence also marks the limits of what can be known. Database reads, private-documentation reads, and direct storage reads were not logged, and one worker server showed outbound traffic of 0.47 MB/s against a usual maximum of 0.29 MB/s [1]. Public reporting does not make clear whether this worker is one of the six affected web servers or a separate system. If it is separate, it is an additional, unexplained signal that falls outside the confirmed exploitation scope.

Downstream Exposure in Application Code

The gadget libraries are dependencies of many Elixir applications, not only of hexpm. The investigation produced coordinated fixes. `db_connection` stops applying `:log` callbacks taken from stream structs (CVE-2026-133555, fixed in 2.10.3). Hex.pm also found the same bug class in `ash` (CVE-2026-124283, fixed in 3.38.0) and reported it to the Ash maintainers [1][3]. The Hacker News reports a related fix in `flop` (CVE-2026-130411) [3]. Further hardening is planned for `plug_crypto`, `plug`, and `ecto` [1]. Any Phoenix or Plug application that decodes externally supplied Erlang terms, even in "safe" mode, should be considered potentially exposed to the same class of attack.

Relevance to AI Development Pipelines

The Elixir ecosystem includes machine learning libraries, and the BEAM runtime is often chosen for real-time and concurrent services. More broadly, AI coding agents and automated dependency tools can resolve and install packages without a human reviewing each step. Pipelines that let an agent add or upgrade dependencies, or that rebuild environments without committed lockfiles, rely heavily on registry metadata integrity. That integrity is exactly what is uncertain while the signing key remains unrotated. The attacker's offline rehearsal against a local copy of open-source server code is also worth noting [1]. Publicly available registry code lets defenders audit it, and it equally lets attackers practice against it. There is no public evidence on whether automated or AI-assisted tooling helped the attacker discover

this gadget chain. In CSA's assessment, organizations should nevertheless plan on the assumption that exploit development against open-source infrastructure will continue to get faster, rather than relying on the time attackers need to study public code.

Recommendations

Immediate Actions

Every individual with a Hex.pm account should change their password and re-enroll 2FA. To re-enroll, go to the account security dashboard, choose to set up a new authenticator app, verify it, and store the new recovery codes [1]. This matters because 2FA secrets may have been read, so the old authenticator entry should be treated as compromised. Sessions and client tokens have already been revoked, so developers and CI systems will need to re-authenticate [1]. Hex.pm advises that existing API keys do not need rotation because of how they are stored [1]. Organizations that want additional assurance may still choose to rotate publishing keys used in CI. Maintainers should also expect targeted phishing that uses leaked email addresses and incident-themed lures. They should check that links point to `hex.pm` and forward suspicious messages to `security@hex.pm` [1].

Build and platform teams should confirm that all Hex traffic goes to `repo.hex.pm` and not to third-party or unofficial mirrors until the new client and signing system are released [1]. Where an internal caching proxy is used, verify that it fetches from the official repository and has not cached registry metadata from untrusted sources. Operators of self-hosted hexpm instances should deploy a build that includes commit `379b0a2` immediately [3][5]. Application teams should update `db_connection`, `ash`, and `flop` to the fixed versions where they are in use [1][3].

Short-Term Mitigations

When Hex.pm releases new clients that support key rotation, teams should update `mix` with `mix local.hex --force` and update `rebar3` and Gleam to match [1]. These updates should go into CI base images and developer onboarding scripts as well as individual workstations, because stale clients are where an old key stays trusted. Teams should also check that `mix.lock` and `rebar.lock` files are committed and enforced in CI. When a project already has a lockfile, fetching dependencies verifies tarball checksums against the locked values [7]. Builds that resolve dependencies freshly are the most exposed to forged metadata, and committed lockfiles shrink that exposure to deliberate upgrades.

Dependency upgrades merged since November 23, 2026, and any made before patched clients with the new signing key are deployed, should get closer review. This is especially true for upgrades made by automated tools or AI coding agents. Teams should compare resolved tarball checksums against values fetched directly from `repo.hex.pm`. Elixir and Erlang application owners should search their code for any decoding of externally supplied Erlang External Term Format data, including `binary_to_term` and `Plug.Crypto.non_executable_binary_to_term`. Any such path should be limited to authenticated, trusted inputs or should explicitly reject structs.

Strategic Considerations

Registry operators, and the foundations and funders that support them, should treat signing-key architecture as part of the supply chain threat model. Registry signing keys should not be readable from internet-facing application servers. They belong in offline or hardware-backed signing services, with key rotation and revocation designed into the clients from the start rather than added after a compromise. This incident shows that a third-party audit, however valuable, is a point-in-time control. It should be combined with continuous practices such as variant analysis for deserialization classes, fuzzing of content-type parsers that run before authentication, and threat modeling of error-handling paths.

Enterprises that depend on open-source ecosystems should add registry-infrastructure compromise to their supply chain incident playbooks alongside malicious-package and maintainer-takeover scenarios. That means keeping an inventory of which registries and mirrors each build system trusts and how to cut traffic over to official sources quickly. It also means knowing which client versions pin which trust anchors. Hex.pm's postmortem, with its minute-by-minute timeline, explicit statement of limits, and same-day regulatory notifications, is a useful benchmark for transparency, while its acknowledged logging gaps show what suppliers should instrument. Organizations can use it to judge their own suppliers' incident transparency and to set expectations in vendor and open-source risk assessments.

CSA Resource Alignment

CSA's whitepaper *The Developer Toolchain as Enterprise Attack Surface* frames IDEs, package registries, and CI/CD systems as one systemic risk surface [12]. The Hex.pm incident is a concrete case of the registry layer described there. It shows that a compromise of registry infrastructure can create ecosystem-wide trust uncertainty even when no package is altered. The whitepaper's emphasis on knowing which upstream trust anchors an enterprise depends on applies directly to the mirror and signing-key guidance above.

The research note *Single Points of Failure: A Week of Supply Chain Compromises* examined how npm, RubyGems, and artifact-repository incidents expose shared control points across the software supply chain [13]. The Hex.pm registry signing key is a clear example of such a control point: one private key whose exposure affects every consumer. Similarly, CSA's analysis of the *RubyGems.org CDN Flaw Exposed Legacy API Keys* incident documented a registry-operator-side weakness rather than a malicious package [14]. Together with this note, it suggests that registry operators themselves deserve a larger share of supply chain threat modeling. CSA's *AI Package Registry Crisis: Unguarded Critical Infrastructure* makes the broader version of this argument, framing package registries as critical infrastructure whose protection has not kept pace with their importance, including for AI development stacks [16]. *Miasma: Cross-Registry Supply Chain Credential Harvesting* covers the complementary threat of credential theft that spreads between ecosystems [11]. Its guidance on credential rotation and phishing resistance applies to Hex.pm maintainers whose password hashes and 2FA secrets may have been exposed.

The CSA AI Controls Matrix (AICM) v1.1 provides the control framework for operationalizing these recommendations [15]. Its Supply Chain Management, Transparency, and Accountability controls cover supplier and open-source dependency risk, including registry trust. Its Cryptography, Encryption, and Key Management domain covers protection and rotation of signing keys. Its Threat and Vulnerability Management and Application and Interface Security domains cover deserialization flaws, variant analysis, and patching of the affected libraries.

References

[1] Ullrich, P., Männchen, J., and Meadows-Jönsson, E. "Security incident on hex.pm." Hex.pm Blog, November 24, 2026.

[2] Erlang Ecosystem Foundation CNA. "CVE-2026-129514." EEF CNA, November 24, 2026.

[3] Khandelwal, S. "Hackers Exploited CVSS 10.0 Hex.pm RCE Flaw to Compromise 6 Package Registry Servers." The Hacker News, November 25, 2026.

[4] Cyber Kendra. "Hex.pm Breach: CVE-2026-129514 Exploited, Reset Your 2FA." Cyber Kendra, November 24, 2026.

[5] hexpm maintainers. "hexpm commit 379b0a2 (fix for CVE-2026-129514)." GitHub, November 2026.

[6] Hex.pm. "Public Keys." Hex.pm Documentation, accessed November 25, 2026.

[7] Hex.pm. "FAQ." Hex.pm Documentation, accessed November 25, 2026.

[8] Hex.pm. "Mirrors." Hex.pm Documentation, accessed November 25, 2026.

[9] Meadows-Jönsson, E. and Männchen, J. "Hex Security Audit." Hex.pm Blog, April 8, 2026.

[10] Coopersmith, A. "Security Audit of Hex, the Erlang package manager." oss-security mailing list, April 12, 2026.

[11] Cloud Security Alliance. "Miasma: Cross-Registry Supply Chain Credential Harvesting." CSA Labs, June 27, 2026.

[12] Cloud Security Alliance. "The Developer Toolchain as Enterprise Attack Surface." CSA Labs, May 25, 2026.

[13] Cloud Security Alliance. "Single Points of Failure: A Week of Supply Chain Compromises." CSA Labs, August 5, 2026.

[14] Cloud Security Alliance. "RubyGems.org CDN Flaw Exposed Legacy API Keys." CSA Labs, July 27, 2026.

[15] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.1." Cloud Security Alliance, 2026.

[16] Cloud Security Alliance. "AI Package Registry Crisis: Unguarded Critical Infrastructure." CSA Labs, June 23, 2026.