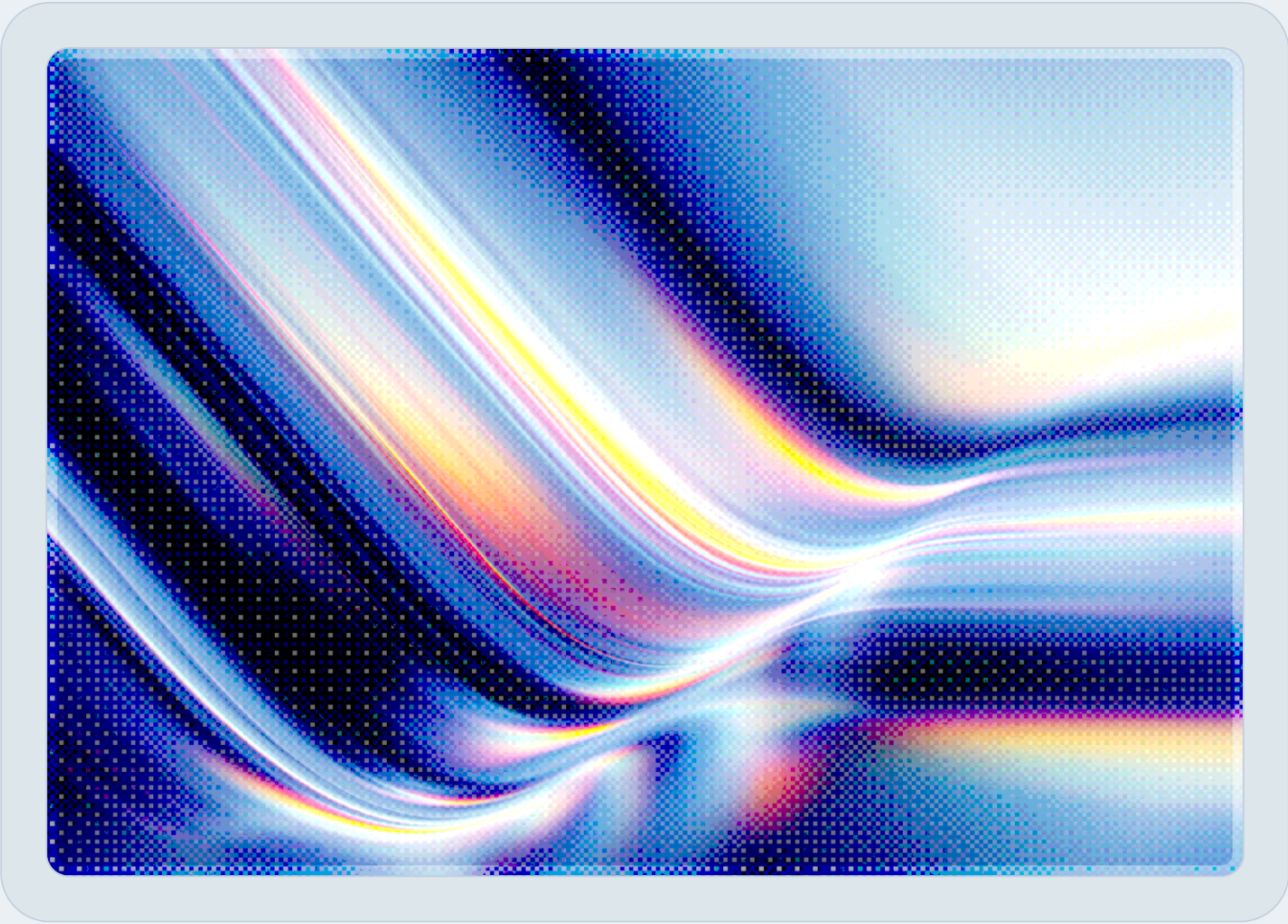
CSAI Foundation | Cloud Security Alliance

# Java4Shell: OpenJDK Base64 Flaw Threatens Identity and DevOps Platforms

## CVE-2026-114938 Security Implications and Guidance

2026-11-19

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

Java4Shell (CVE-2026-114938) is a memory-corruption flaw in the hand-written x86-64 assembly that the OpenJDK HotSpot virtual machine uses to accelerate Base64 decoding on processors with AVX-512 VBMI instructions. A malformed Base64 string can make the decoder write beyond the end of its output buffer before Java rejects the input [1][2]. Security firm Calif disclosed the flaw on November 17, 2026, and Oracle and the OpenJDK Vulnerability Group released fixed builds the same day [1][3]. Calif reports that it achieved pre-authentication remote code execution against default installations of Keycloak, Atlassian Jira and Confluence, JFrog Artifactory, Jenkins, and PingFederate [1][3]. Those products sit at the center of enterprise identity and software delivery, which makes this a priority for any organization running them on affected hardware.

The most important point for defenders is that the fix lives in the Java runtime, not in the applications. Fixed builds are Oracle and OpenJDK 17.0.21.1, 21.0.13.1, 25.0.5.1, and 27.0.1.1, along with corresponding Corretto, Zulu, and GraalVM builds. Oracle's Java 8 Performance Pack is also affected and is fixed in 8u513-perf, while standard Java 8 and Java 11 builds are not affected [3][6]. Products and container images that bundle their own JRE therefore need a vendor update or a rebuild; patching the host's system JDK is not enough [3][4].

Exposure also depends on hardware. Only x86-64 hosts with AVX-512 VBMI run the vulnerable code path, including Intel Xeon Ice Lake and later and AMD Zen 4 and later, and ARM64 is not affected [1][4]. As a result, the same container image can be vulnerable on one node and safe on another, and the CPU flag check described under Recommendations is the authoritative test for any given host. Where patching must wait, starting the JVM with `-XX:-UseBASE64Intrinsics` disables the vulnerable code path. Neither the Oracle nor the OpenJDK advisory mentions this flag, but Calif recommends it and Keycloak endorses it as an interim measure [1][3][14].

Defenders should assume that denial of service is easy and that remote code execution is hard but credible. Independent labs have publicly reproduced crashes and memory disclosure. Wiz, VulnCheck, and an Immersive researcher report private remote code execution exploits that have not been publicly demonstrated or independently verified [2][3]. GreyNoise has observed exploitation attempts, but as of November 18 no successful in-the-wild exploitation had been confirmed, and the CVE was not in CISA's Known Exploited Vulnerabilities (KEV) catalog [3][4].

Finally, the case illustrates both AI-assisted discovery and slow triage. Calif says a researcher using an AI coding agent found the flaw in about an hour and built a first exploit in two more, for under $200 in total token costs [1]. Separately, a trigger for an overflow in the same code sat in an unmerged OpenJDK pull request for two months [4]. Together, these facts are consistent with CSA's assessment that the bottleneck is shifting from finding flaws to remediating them.

# Background

Base64 decoding is one of the most common operations in Java server software. HTTP Basic authentication headers, SAML artifacts, JSON Web Token segments, and CSRF tokens are Base64-encoded, and servers usually decode them before deciding whether a request is authenticated [3][4][7]. For performance, HotSpot replaces frequently executed Java library methods with "intrinsics," hand-tuned machine code that the just-in-time (JIT) compiler substitutes once a method becomes hot. HotSpot includes an intrinsic for `java.util.Base64` decoding that uses AVX-512 VBMI vector instructions on supporting Intel and AMD processors, and Calif reports that the vulnerable version is present in OpenJDK 17.0.6 and later [1][5]. Because the intrinsic only engages after JIT compilation on specific hardware, the vulnerable code does not appear in source-level code review of the Java libraries. Software composition analysis tools that flag application dependencies also miss it.

Calif's technical write-up places the root cause in the handling of trailing padding. When input ends in an `=` character, the intrinsic decrements an internal output length below zero. The underflowed value produces an all-ones write mask, so a vector store meant to write a few bytes writes a full 64-byte block [1]. In Calif's example, a 65-character string (64 `A` characters followed by `=`) aimed at a 50-byte buffer writes 62 bytes past the buffer's end; on OpenJDK 17, the trigger needs 128 `A` characters before the padding [1]. The decoder then correctly throws an `IllegalArgumentException`, but only after the out-of-bounds write has happened [1][4]. Because the overwritten bytes are largely fixed, an attacker controls where the corruption lands, not what it contains. That limitation drives much of the exploitation difficulty discussed below.

The disclosure did not follow a clean embargo. According to The Hacker News, a malformed input that triggers an overflow appeared publicly on September 9, 2026. It was posted in an OpenJDK pull request (#32781) proposing a fix for an overflow in the same AVX-512 Base64 decoder code, and that pull request was closed without merging in early November after an automated inactivity notice [4]. Available reporting does not establish whether the pull request concerned the same root cause as CVE-2026-114938. Tenable reports that Calif announced the flaw on November 17 after details leaked before a fix was available. The same day, Oracle published its November 2026 Critical Patch Update, the OpenJDK

Vulnerability Group published its advisory, and CISA and the Canadian Centre for Cyber Security issued alerts [3][6]. Calif credits researcher Quang Luong with the discovery, and Oracle credits Luong and Albert Yang [1][3].

Severity scores vary across sources, apparently reflecting different scoring versions and different judgments of attack complexity and required privileges rather than disagreement about the flaw itself. The table below summarizes the published ratings.

| Source | Score | Notes |
|---|---|---|
| Calif (discoverer) | 9.2 (CVSS 4.0) | Unauthenticated network attack vector [1][5] |
| Oracle / OpenJDK | 8.1 (CVSS 3.1) | Attack complexity rated High; no authentication required [3] |
| Red Hat | 9.8 (preliminary; version not stated) | No authentication required [4] |
| Amazon | 8.8 (version not stated) | Crash for unauthenticated users; code execution by authenticated low-privilege users in certain environments [4] |
| Canadian Cyber Centre | Critical | Classified as CWE-122 heap-based buffer overflow [6] |

# Security Analysis

## Why identity providers and DevOps platforms are the priority

The products Calif named share a common trait: they decode attacker-supplied Base64 before authentication. Identity providers such as Keycloak and PingFederate decode Base64-encoded SAML messages and credentials on their login pages, which are by design reachable by unauthenticated users. Atlassian, Jenkins, and Artifactory deployments accept HTTP Basic authentication and API tokens at their front doors [1][4]. Calif reports pre-authentication code execution against default configurations of Keycloak, Jira, Confluence, Artifactory, Jenkins, and PingFederate. It also reports post-authentication

code execution against Apache Kafka brokers, Apache Solr, Elasticsearch, Gerrit, and Spinnaker, notes that the list is not exhaustive, and says it achieved remote code execution against sample applications built on Apache Struts and Spring Boot [1][3].

The business impact of a compromise is disproportionate in both categories. Code execution on an identity provider can expose signing keys and session secrets, potentially allowing an attacker to forge authentication to federated applications that trust those keys. Code execution on a CI/CD server or artifact repository gives an attacker a position to tamper with builds and published artifacts, turning a single host compromise into a software supply chain incident. Calif also describes a memory-disclosure stage it calls "JavaBleed" that may expose keys and tokens even when full code execution is not achieved [3]. Blackpoint Cyber reported that its local test of this read primitive recovered a secret placed next to the overflowed buffer in all 30 runs [4]; that was a controlled setup in which the secret was deliberately positioned, and real-world reliability will depend on heap layout. Even so, organizations should treat unpatched, exposed identity and DevOps systems as candidates for credential rotation, not only for patching.

Calif further reports a client-side scenario in which a Java program that opens a TLS connection to an attacker-controlled server can be compromised. The exploit for this scenario came from researcher samczsun, and Calif has not published how it reaches the decoder [3][4]. If confirmed, this would extend exposure to Java services and agents that make outbound connections, including build agents that fetch dependencies and integration services that call third-party APIs. Such systems could be at risk even if they expose no listening port.

## Exploitability: easy to crash, hard to weaponize

Reports so far support a nuanced risk picture. On the denial-of-service side, the crash has been independently reproduced. Blackpoint Cyber crashed a test Spring Boot application with roughly 28,300 unauthenticated malformed login requests over about 20 seconds, a volume that is trivial for an attacker to generate, and Upwind reproduced memory corruption leading to a crash on an AWS host with an Intel Ice Lake processor [4]. OX Security likewise observes that a short burst of requests can reliably bring an affected service down [7]. Because it is difficult to confirm from the outside whether a given endpoint passes untrusted input to the vulnerable decoder, organizations should treat any reachable, unpatched Java service on affected hardware as potentially vulnerable to remote crashes now.

Remote code execution is considerably harder. Calif's write-up describes turning the fixed-value overwrite into a bounded write by corrupting an adjacent array's length, then building a read primitive the same way and using heap grooming and network timing to place objects. Calif notes that reliability depended on low, stable round-trip times and that it ran attacks from a virtual machine in the same cloud region as the target [1]. It states that each target requires a bespoke chain and is withholding application-

specific exploits until users have had time to upgrade [1]. OX Security models the cumulative conditions an attack must satisfy: an affected Java version, an unpatched runtime, an x86-64 host with AVX-512 VBMI, a JIT-compiled decoder, untrusted input reaching the affected decoder before authentication, internet exposure, and a feasible tailored exploit. On that basis it estimates that under 0.12% of JVMs are realistic remote code execution targets and roughly 0.8% are remotely crashable [7]. OX acknowledges that these are approximate estimates built on assumed independence between conditions. A small percentage of a very large installed base may still represent many high-value systems, especially given the concentration in identity and DevOps tooling.

The threat picture is moving quickly. GreyNoise has reported exploitation attempts and scanning, although no confirmed successful exploitation on its sensors. Wiz states that it proved remote exploitation is feasible against standard Java applications such as Jenkins, Confluence, and Apache Tomcat, and that its AI harness built a proof-of-concept exploit within two hours [2]. VulnCheck says it has a working exploit available to its customers, and Kevin Breen of Immersive says he developed an application-specific exploit that he has not released [3]. None of these private exploits had been publicly demonstrated or independently verified at the time of writing. As of November 17, the Canadian Cyber Centre was not aware of active exploitation but warned that this could change quickly [6]. Given that private exploits are reportedly already in hand, that a trigger input has been public since September, and that Calif's exploitation methodology is documented at a conceptual level, CSA assesses that weaponized exploits for the most common targets are likely to become publicly or criminally available. That assessment is an inference, not a confirmed development.

## Why inventory is harder than usual

Java4Shell defeats several assumptions that vulnerability management programs rely on. First, the vulnerable component is the runtime, which many enterprise products ship privately inside their installation directories or container images, so patching the host's system JDK leaves those bundled copies untouched [4][6]. Second, exposure depends on the physical CPU, which means cloud instance type selection, autoscaling, and Kubernetes scheduling decide whether a given workload is vulnerable at a given moment [4]. Third, downstream distribution timing varies widely. As of November 18, Red Hat (RHSB-2026-018) and Ubuntu had not released fixes, and Debian had fixed only its unstable branch, while Amazon Corretto, Azul Zulu, and Oracle builds were available and Eclipse Temurin builds were rolling out [3][4].

The table below summarizes fixed builds reported at the time of writing. Because distribution status for Red Hat, Ubuntu, and Debian is likely to change within days, readers should verify against current vendor advisories before acting on it.

| Distribution | Fixed builds reported (as of Nov 18, 2026) |
|---|---|
| Oracle Java SE / OpenJDK | 17.0.21.1, 21.0.13.1, 25.0.5.1, 27.0.1.1; Java 8 Performance Pack 8u513-perf [3][6] |
| Amazon Corretto | 17.0.21.1.2, 21.0.13.1.2, 25.0.5.1.2, 27.0.1.1.4 [4] |
| Azul Zulu (free builds) | 17.70.201, 21.54.201, 25.38.201, 27.30.201 [4] |
| GraalVM | GraalVM for JDK 17.0.21.1 and 21.0.13.1; GraalVM Enterprise 21.3.20.1 [3][6] |
| Eclipse Temurin | Fixed builds rolling out [3] |
| Red Hat / Ubuntu / Debian stable | Fix pending at time of reporting; verify current vendor advisories [3][4] |

Standard Java 8 and Java 11 builds do not contain the vulnerable intrinsic and are not listed as affected, and OpenJDK 17.0.0 through 17.0.5 predate it [3][5]. Organizations should not read this as a reason to keep end-of-life runtimes, which carry their own unpatched vulnerabilities.

## Detection

Detection is possible but imperfect. Upwind suggests looking for Base64 tokens of 32 or more characters that end in `=` and whose length is not a multiple of four, particularly in `Authorization` headers. It also points to bursts of `IllegalArgumentException` errors from `java.util.Base64$Decoder`, and to unexplained `hs_err_pid` JVM crash logs or restart loops [4]. Upwind cautions, however, that none of these indicators proves compromise and that their absence does not prove safety [4]. Calif also reports that request buffering in a reverse proxy such as nginx disrupts some exploitation chains that rely on network timing, but not all; the Elasticsearch chain still works through it [1][4].

# Recommendations

## Immediate Actions

Organizations should first identify every Java runtime that could process untrusted input, recording vendor and full build version as the Canadian Cyber Centre recommends [6]. This inventory must include JREs bundled inside commercial products, appliances, and container images, not only system packages. Patching should start with internet-facing identity providers (Keycloak, PingFederate, and any Java-based SSO), then move to source control, CI/CD, and artifact systems (Jenkins, Gerrit, Artifactory, Atlassian products). Each patched service must be restarted, and container images must be rebuilt and redeployed, because a fixed runtime on disk does nothing for a running JVM.

Where a fixed runtime is not yet available from the relevant vendor, administrators should add `-XX:-UseBASE64Intrinsics` to the JVM options. This disables only the vulnerable fast path, and Upwind reports that it stopped the overflow in testing [1][4]. The alternative flag `-XX:UseAVX=2`, which the Canadian Cyber Centre also recommends, stops the overflow but disables all AVX-512 optimizations [1][3][6]. It must not be applied to ARM64 hosts, where the JVM will fail to start; since ARM64 is unaffected, this matters mainly for teams that apply JVM options uniformly across mixed-architecture fleets [1][3]. Upwind warns that `-XX:DisableIntrinsic=_base64_decodeBlock` used alone prevents Java from starting [4]. Teams should test flag changes on representative hardware and track them so they can be removed after patching. To scope exposure on Linux hosts, confirm whether `avx512f`, `avx512bw`, `avx512vbmi`, and `bmi2` all appear in `/proc/cpuinfo` [3].

## Short-Term Mitigations

Within the following days, organizations should hunt for the indicators described above across web server, application, and authentication logs, and treat clusters of JVM crashes on exposed services as potential exploitation attempts [4][6]. For identity providers and CI/CD systems that were internet-exposed and unpatched on affected hardware, a precautionary rotation of signing keys, client secrets, service tokens, and deployment credentials is proportionate given the documented memory-disclosure primitive [3][4]. Where feasible, administrative and login interfaces of DevOps platforms should be moved behind a VPN, identity-aware proxy, or IP allow list. That step removes the path from unauthenticated internet hosts and narrows the population that can reach the vulnerable decoder, although users who pass those controls, compromised internal hosts, and allow-listed sources can still reach it. Teams should also ask each software vendor whether its product bundles an affected runtime and when a fixed build will ship, since vendor timelines are likely to lag the upstream fix.

### Strategic Considerations

Java4Shell exposes a gap in how most programs track exposure. In CSA's assessment, few asset inventories or SBOMs record which JVM build a product embeds, and fewer still correlate software with CPU instruction-set features, because SBOMs describe build-time composition while CPU features are a deployment-time property. Organizations should extend SBOM and runtime inventory practices to cover embedded runtimes and treat hardware characteristics as an exposure dimension for low-level platform flaws. They should also favor architectures in which runtimes are patched once and rebuilt everywhere, through immutable images, centralized base images, and fast redeploy pipelines, over long-lived hosts patched in place.

The way the flaw was found matters as much as the flaw itself. A single researcher with an AI coding agent reportedly found a critical flaw in a foundational runtime in about an hour [1]. Meanwhile, a public trigger for an overflow in the same code sat in an unmerged OpenJDK pull request for roughly two months [4]. Cases like this suggest the binding constraint is shifting from discovery toward triage, coordinated fixing, and downstream distribution. Given the falling cost of AI-assisted discovery [8][9], security leaders should plan for more frequent, high-severity flaws in foundational runtimes. They should invest in remediation throughput and in compensating controls, such as segmentation, least privilege, and behavior-based detection, that limit damage while patches propagate.

## CSA Resource Alignment

CSA's November 2026 whitepaper *The Disclosure Bottleneck: When AI Discovery Outpaces Patch Consumption* [8] is the most directly relevant prior work. It argues that frontier AI has made finding vulnerabilities cheap, which shifts constraints to maintainer triage, fix development, and downstream packaging. It also observes that a patch being released upstream does not mean the patch is widely installed. Java4Shell is consistent with each stage of that argument: a reportedly AI-assisted discovery, a public trigger that sat in an unmerged upstream pull request, and an upstream fix that, as of November 18, had not yet reached Red Hat, Ubuntu, or Debian stable builds. The paper's recommendations to measure patch consumption capacity, maintain accurate SBOMs, and reduce reliance on any single patch map directly to the embedded-runtime and rebuild guidance above.

The June 2026 whitepaper *AI-Accelerated Vulnerability Discovery and the Patch Debt Crisis* [9] provides the strategic framing for prioritization. It calls for a two-tier, risk-based model with rapid remediation for internet-facing systems that face known or likely exploitation, and for architectural investments such as segmentation and immutable infrastructure. Java4Shell's split between easy denial of service and hard but plausible remote code execution, concentrated in identity and DevOps systems, illustrates why CVSS

alone is a poor sorting key and why exposure and asset criticality must drive sequencing. The joint SANS, CSA, [un]prompted, and OWASP GenAI "AI Vulnerability Storm" strategy briefing [10] offers CISO-level guidance on operating as exploit timelines compress.

For control mapping, the CSA AI Controls Matrix (AICM) v1.1 [11] provides the relevant requirements in its Threat and Vulnerability Management (TVM) domain, covering vulnerability identification, prioritization, and remediation, and in its Application and Interface Security (AIS) domain. Organizations operating AI services that depend on Java-based identity or pipeline components can use the *AICM v1.1 Implementation Guidelines for Application Providers* [12] to map Java4Shell response activities to auditable controls. Finally, CSA's *Zero Trust Guiding Principles* [13] support the short-term recommendation to remove pre-authentication network reachability from administrative and DevOps interfaces. Under zero trust, a login page that any internet host can reach is an exposure to minimize, not a given.

# References


[1] Calif. "Java4Shell: Remote Code Execution in OpenJDK." Calif, November 17, 2026.

[2] Schwartz, T., and Bar, M. "CVE-2026-114938 OpenJDK RCE (Java4Shell): Everything You Need to Know." Wiz Blog, November 17, 2026.

[3] Narang, S. "Java4Shell (CVE-2026-114938): Frequently Asked Questions About OpenJDK RCE Vulnerability." Tenable Blog, November 17, 2026 (updated November 18, 2026).

[4] Khandelwal, S. "Java4Shell Flaw (CVE-2026-114938) Enables Pre-Auth RCE in Jira, Jenkins, Keycloak and More." The Hacker News, November 18, 2026.

[5] Semgrep. "Java4Shell: A Critical Java Vulnerability Hiding in Plain Sight." Semgrep Blog, November 2026.

[6] Canadian Centre for Cyber Security. "AL26-029 – Critical Vulnerability Impacting Oracle Java – CVE-2026-114938." Government of Canada, November 17, 2026.

[7] Siman Tov Bustan, M., and Zadok, N. "Java4Shell (CVE-2026-114938): Easy to Crash, Hard to Exploit – Less Than 0.12% of JVMs Estimated at Risk of RCE." OX Security Blog, November 18, 2026.

[8] Cloud Security Alliance AI Safety Initiative. "The Disclosure Bottleneck: When AI Discovery Outpaces Patch Consumption." Cloud Security Alliance, November 6, 2026.

[9] Cloud Security Alliance AI Safety Initiative. "AI-Accelerated Vulnerability Discovery and the Patch Debt Crisis." Cloud Security Alliance, June 17, 2026.

[10] Cloud Security Alliance. "SANS Institute, Cloud Security Alliance, [un]prompted, and OWASP GenAI Security Project Release Emergency Strategy Briefing as AI-Driven Vulnerability Discovery Compresses Exploit Timelines from Weeks to Hours." Cloud Security Alliance, April 14, 2026.

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

[12] Cloud Security Alliance. "AICM v1.1 Implementation Guidelines for Application Providers (AP)." Cloud Security Alliance, 2026.

[13] Cloud Security Alliance. "Zero Trust Guiding Principles." Cloud Security Alliance.

[14] Keycloak. "Java4Shell: What Keycloak admins need to do." Keycloak Blog, November 18, 2026.