INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-12 14:08 UTC

Azure SDK for Java Security Feature Bypass Vulnerability (CVE-2026-33117)

CVE VULNERABILITY | CRITICAL | CVSS 9.1

SCC Item ID	SCC-CVE-2026-0159
Type	CVE Vulnerability
CVE ID	CVE-2026-33117
Severity	CRITICAL
CVSS Base Score	9.1
Affected Products	Microsoft Azure SDK for Java
Published	2026-05-12T07:00:00
Discovery Source	Msrc Patch Tuesday

Executive Summary

Microsoft disclosed a critical security feature bypass vulnerability (CVE-2026-33117, CVSS 9.1) in the Azure SDK for Java as part of the May 2026 Patch Tuesday release. Organizations using the Azure SDK for Java in applications that authenticate or authorize access to Azure resources and services are directly affected. If exploited, this vulnerability could allow an attacker to bypass authentication or authorization controls enforced by the SDK, enabling unauthorized access to Azure-hosted data, services, or infrastructure.

Technical Analysis

CVE-2026-33117 is a security feature bypass vulnerability in the Microsoft Azure SDK for Java, disclosed on May 2026 Patch Tuesday. It carries a CVSS base score of 9.1 (Critical) and is classified under CWE-284 (Improper Access Control) and CWE-287 (Improper Authentication). The vulnerability maps to MITRE ATT&CK techniques T1078 (Valid Accounts), T1550 (Use Alternate Authentication Material), and T1190 (Exploit Public-Facing Application), indicating the bypass likely circumvents authentication or token-validation logic within the SDK. Specific affected SDK versions and the precise bypass mechanism will be specified in the MSRC advisory at https://msrc.microsoft.com/update-guide/vulnerability/CVE-2026-33117 and the NVD entry at https://nvd.nist.gov/vuln/detail/CVE-2026-33117. No CISA KEV listing is present as of this writing, and EPSS data was not available. No known threat actor exploitation has been reported at this time.

Action Checklist

1. Step 1: Containment, Identify all applications and services in your environment that import or depend on the Microsoft Azure SDK for Java. Restrict or isolate those workloads from internet-facing exposure until the patch status is confirmed. Consult the MSRC advisory (https://msrc.microsoft.com/update-guide/vulnerability/CVE-2026-33117) to determine affected version ranges; do not assume all versions are affected without verification.
2. Step 2: Detection, Query your software inventory, SBOM records, and dependency manifests (pom.xml, build.gradle) for references to com.azure or azure-sdk-for-java packages. Check Azure Monitor, Entra ID (formerly Azure AD) sign-in logs, and application logs for anomalous authentication patterns: unexpected valid-account sign-ins, token reuse across sessions, or access from unexpected source IPs against Azure resources served by SDK-dependent applications. Correlate against T1078 and T1550 detection logic in your SIEM.
3. Step 3: Eradication, Apply the patched Azure SDK for Java version identified in the MSRC advisory. Update dependency declarations in all affected build files and rebuild application artifacts. Rotate any Azure service principal credentials, managed identity tokens, or access keys used by applications running the vulnerable SDK version, as a precaution against possible prior exploitation.
4. Step 4: Recovery, After deploying the patched SDK version, confirm the updated dependency is present in deployed artifacts via your artifact registry or container image scanning tool. Monitor Azure sign-in and resource access logs for a minimum of 72 hours post-remediation for residual anomalous access. Re-run your dependency scanner to confirm no remaining references to the vulnerable version.
5. Step 5: Post-Incident, Review your SBOM and dependency management process: this vulnerability highlights the risk of SDK-layer authentication bypass in cloud-native applications. Evaluate whether your CI/CD pipeline includes automated dependency vulnerability scanning (e.g., Dependabot, OWASP Dependency-Check) that would flag such issues before deployment. Map the control gap to NIST SP 800-53 SI-2 (Flaw Remediation) and SA-15 (Development Process, Standards, and Tools) for GRC tracking.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate immediately to CISO and legal/compliance if Entra ID sign-in logs show successful authentication events (`ResultType=0`) from the vulnerable application's service principal or managed identity originating from IPs outside approved egress ranges during the exploitation window, as this constitutes potential unauthorized access to Azure-hosted data that may trigger breach notification obligations under GDPR, CCPA, or HIPAA depending on the data classification of the Azure resources accessed by the affected SDK-dependent applications.

Recovery Notes	After deploying the patched Azure SDK for Java version confirmed in the MSRC advisory for CVE-2026-33117, verify recovery by running Trivy or OWASP Dependency-Check against all rebuilt artifacts and confirming zero findings for the vulnerable com.azure package coordinates before restoring internet-facing exposure to the previously isolated workloads. Maintain continuous monitoring of Entra ID sign-in logs and Azure Monitor resource access logs for the remediated service principal and managed identity ObjectIDs for a minimum of 72 hours post-patch, extending to 7 days if any anomalous access was detected during the initial detection phase. If credential rotation was performed, additionally verify that no Azure resource access events succeed using the revoked credentials during this window, which would indicate a persistent token or credential was captured by an attacker prior to rotation.
Forensic Artifacts	Entra ID Sign-in Logs (Azure Portal → Entra ID → Monitoring → Sign-in logs): Filter on the service principal ClientIDs of SDK-dependent applications for the 30-day pre-discovery window; a CVE-2026-33117 bypass would appear as ResultType=0 (success) with ConditionalAccessStatus=notApplied or notEnabled, indicating authentication controls enforced via the SDK were circumvented without triggering policy blocks. | Azure Monitor Diagnostic Logs for affected resources (Storage, Key Vault, Service Bus, etc.): Resource access events logged under the managed identity or service principal ObjectID of the vulnerable application — exploitation of an SDK-layer auth bypass would produce authorized-looking access events that correlate with anomalous source IPs or off-hours access patterns not attributable to normal application behavior. | Application-tier com.azure.identity and com.azure.core library debug logs: If the application's logging framework (Log4j2, SLF4J, Logback) was configured at DEBUG level for the com.azure namespace, these logs capture token acquisition flows and HTTP pipeline events — a bypass exploit would produce anomalous token grant sequences where expected authentication challenges (MFA, conditional access) are absent from the logged exchange. | Azure SDK for Java local token cache artifacts: The Azure Identity SDK may persist MSAL token cache to disk at `~/.azure/msal_token_cache.json` (Linux) or `%USERPROFILE%\.azure\msal_token_cache.json` (Windows) depending on application configuration — capture and preserve this file pre-eradication to analyze whether tokens acquired during a potential exploitation window remain cached and could be replayed. | Dependency manifest files and build artifacts (pom.xml, build.gradle, compiled JAR/WAR MANIFEST.MF): These establish the definitive record of which com.azure SDK version was deployed at time of incident, including transitive dependency resolution — run `sha256sum` on all deployed JARs containing com.azure classes and preserve against the Maven Central published hash for the identified vulnerable version to confirm exploit-eligible code was present in the production deployment.

Per-Action IR Details

Step 1: Containment — Identify all applications and services in your environment that import or depend on the Microsoft Azure SDK for Java. Restrict or isolate those workloads from internet-facing exposure until the patch status is confirmed. Consult the MSRC advisory (https://msrc.microsoft.com/update-guide/vulnerability/CVE-2026-33117) to determine affected version ranges; do not assume all versions are affected without verification.

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment Strategy

Controls: NIST IR-4 (Incident Handling), NIST SI-2 (Flaw Remediation), NIST AC-4 (Information Flow Enforcement), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 4.4 (Implement and Manage a Firewall on Servers)

Compensating: Run `mvn dependency:tree | grep -i 'com.azure'` or `grep -r 'com.azure' --include='*.xml' --include='*.gradle' /path/to/repos` across all build workspaces to enumerate affected applications without a centralized

SBOM tool. Use host-based firewall rules (`iptables -I OUTPUT -p tcp --dport 443 -j DROP` scoped to the service account UID) or network ACLs on the perimeter router to block outbound Azure API calls from confirmed-vulnerable application hosts until the patch is validated. For containerized workloads, `docker inspect` each image for the com.azure JAR and pause containers running vulnerable versions with `docker pause `.

Evidence: Before isolating workloads, capture: (1) full outbound network connection state from Azure SDK-dependent JVM processes using `ss -tunap | grep ` or `netstat -anp` — preserving active connections to Azure service endpoints (*.azure.com, *.microsoft.com, login.microsoftonline.com) that may reflect in-progress exploitation; (2) running process list with full command-line arguments (`ps auxww | grep java`) to document which SDK-dependent applications are live at time of containment; (3) application-tier HTTP access logs (Tomcat access log, Spring Boot embedded server logs at default path `/var/log//`) for any anomalous inbound requests to authentication endpoints immediately preceding isolation.

Step 2: Detection — Query your software inventory, SBOM records, and dependency manifests (pom.xml, build.gradle) for references to com.azure or azure-sdk-for-java packages. Check Azure Monitor, Entra ID (formerly Azure AD) sign-in logs, and application logs for anomalous authentication patterns: unexpected valid-account sign-ins, token reuse across sessions, or access from unexpected source IPs against Azure resources served by SDK-dependent applications. Correlate against T1078 and T1550 detection logic in your SIEM.

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 — Detection and Analysis

Controls: NIST SI-4 (System Monitoring), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-2 (Event Logging), NIST IR-5 (Incident Monitoring), CIS 8.2 (Collect Audit Logs), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Without a SIEM, query Entra ID sign-in logs directly via Microsoft Graph CLI: `az monitor activity-log list --start-time 2026-05-01 --query "[?contains(operationName.value, 'signIn')]"` or pull the Entra ID Sign-in log CSV from the Azure Portal (Entra ID → Monitoring → Sign-in logs) and parse with `jq` or PowerShell `Import-Csv` filtering on `ResultType -ne 0` for failures and cross-referencing `IPAddress` against known baselines. For T1550 (Use Alternate Authentication Material) detection specific to SDK-issued tokens, use osquery: `SELECT * FROM process_open_sockets WHERE process_name='java' AND remote_address LIKE '%.azure.com'` combined with manual review of the application's token cache files (Azure Identity SDK caches tokens in memory by default; if the app persists tokens to disk, check `~/.azure/` or `%USERPROFILE%\.azure\` for `msal_token_cache.json`). Write a Sigma rule targeting Entra ID logs for `ResultType=0` (successful sign-in) combined with `ConditionalAccessStatus=notApplied` from source IPs not in your approved egress range.

Evidence: Capture before analysis: (1) Entra ID sign-in logs exported for the 30 days preceding discovery, filtering on the service principal ClientIDs used by SDK-dependent applications — a bypass exploit would manifest as `ResultType=0` sign-ins where conditional access policies show `notApplied` or `notEnabled`, indicating the SDK bypassed enforcement; (2) Azure Monitor Diagnostic Logs for the specific Azure resources (Storage, Key Vault, Service Bus, etc.) accessed by the vulnerable application — look for resource access events from the application's managed identity or service principal during off-hours or from unexpected source IPs; (3) application-level logs for `com.azure.identity` or `com.azure.core.http` library log output (set logger to DEBUG level for `com.azure` namespace and capture to file) showing token acquisition flows that succeed without expected authentication challenges; (4) MITRE ATT&CK T1078 (Valid Accounts) and T1550 (Use Alternate Authentication Material) — specifically look for Azure token reuse indicators in Entra ID logs: same `correlationId` or `sessionId` appearing across multiple distinct `IPAddress` values.

Step 3: Eradication — Apply the patched Azure SDK for Java version identified in the MSRC advisory. Update dependency declarations in all affected build files and rebuild application artifacts. Rotate any Azure service principal credentials, managed identity tokens, or access keys used by applications running the vulnerable SDK version, as a precaution against possible prior exploitation.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST SI-2 (Flaw Remediation), NIST IA-5 (Authenticator Management), NIST AC-2 (Account Management), NIST CM-6 (Configuration Settings), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management), CIS 5.2 (Use Unique Passwords)

Compensating: For teams without an enterprise artifact management pipeline: update `pom.xml` to pin the patched `com.azure:azure-sdk-bom` version (per MSRC advisory) and run `mvn versions:display-dependency-updates` to confirm no transitive dependency pulls the vulnerable version back in. Rebuild with `mvn clean package -U` and verify the output JAR/WAR contains only the patched SDK using `unzip -l target/*.jar | grep azure`. For credential rotation without an enterprise secrets manager: use Azure CLI to rotate service principal secrets (`az ad app credential reset --id --append`), revoke all existing tokens for managed identities via `az identity` commands, and immediately update the new secrets in application configuration — for Key Vault-backed apps, use `az keyvault secret set`. Document all rotated credential IDs with timestamps for the post-incident record.

Evidence: Before executing eradication, preserve: (1) a checksum inventory of all currently deployed JAR files containing com.azure dependencies (`find /opt /srv /app -name '*.jar' -exec sha256sum {} \;`) to establish a pre-patch baseline for later comparison; (2) the full list of Azure service principal `objectId` and `appId` values associated with the vulnerable applications, exported via `az ad sp list --query "[].{id:id,appId:appId,displayName:displayName}"` — these are required to audit post-rotation for any residual credential use; (3) current Azure role assignments for the affected service principals (`az role assignment list --assignee `) to document the blast radius if credentials were compromised during the exploitation window.

Step 4: Recovery — After deploying the patched SDK version, confirm the updated dependency is present in deployed artifacts via your artifact registry or container image scanning tool. Monitor Azure sign-in and resource access logs for a minimum of 72 hours post-remediation for residual anomalous access. Re-run your dependency scanner to confirm no remaining references to the vulnerable version.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST SI-7 (Software, Firmware, and Information Integrity), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-11 (Audit Record Retention), NIST IR-4 (Incident Handling), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 2.1 (Establish and Maintain a Software Inventory)

Compensating: Without a commercial container scanner, use Trivy (free, open-source) to scan rebuilt container images: `trivy image --severity CRITICAL --ignore-unfixed :` — confirm zero findings for CVE-2026-33117 in the com.azure package tree. For non-containerized deployments, use OWASP Dependency-Check CLI (`dependency-check.sh --project myapp --scan /path/to/app --format JSON`) against rebuilt artifacts and validate the JSON report shows no vulnerable `com.azure` coordinates. For the 72-hour monitoring window without SIEM, create a daily cron job that pulls Entra ID sign-in logs via Graph API and pipes them through a PowerShell script checking for the service principal ClientIDs of the previously vulnerable applications: `Get-MgAuditLogSignIn -Filter "appId eq '' | Where-Object {$_.ResultType -ne 0}` — alert on any non-zero ResultType or unexpected source IP.

Evidence: During the recovery monitoring window, collect and preserve: (1) Azure Monitor resource access logs for the 72-hour post-patch window, specifically filtering on the managed identity or service principal ObjectIDs of the remediated applications — any successful resource access from the old, now-rotated credentials would indicate a credential compromise predating rotation; (2) artifact registry or container registry layer diff confirming the patched `com.azure` BOM version is present in the rebuilt image digest, logged with SHA-256 image hash for integrity verification; (3) dependency scanner output report (Trivy or OWASP Dependency-Check JSON) from the post-patch scan, retained as evidence of eradication for GRC audit trail per NIST AU-11 (Audit Record Retention).

Step 5: Post-Incident — Review your SBOM and dependency management process: this vulnerability highlights the risk of SDK-layer authentication bypass in cloud-native applications. Evaluate whether your CI/CD pipeline includes automated dependency vulnerability scanning (e.g., Dependabot, OWASP Dependency-Check) that would gate such issues before deployment. Map the control gap to NIST SP 800-53 SI-2 (Flaw Remediation) and SA-15 (Development Process, Standards, and Tools) for GRC tracking.

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity (Lessons Learned)

Controls: NIST SI-2 (Flaw Remediation), NIST SA-15 (Development Process, Standards, and Tools), NIST IR-8 (Incident Response Plan), NIST RA-3 (Risk Assessment), NIST AU-6 (Audit Record Review, Analysis, and Reporting), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 2.1 (Establish and Maintain a Software Inventory)

Compensating: For teams without enterprise SBOM tooling: implement a mandatory `mvn dependency:tree > sbom-$(date +%F).txt` step as a CI/CD gate in your Jenkins pipeline or GitHub Actions workflow, archiving the output as a build artifact. Add OWASP Dependency-Check as a Maven plugin (`org.owasp:dependency-check-maven`) that fails the build on CVSS ≥ 7.0 findings — this directly addresses the control gap where CVE-2026-33117 would have been caught at build time rather than at advisory disclosure. For GRC documentation, create a finding record mapping: control gap (no automated SDK vulnerability scanning in CI/CD) → NIST SI-2 (Flaw Remediation) deficiency → remediation action (Dependency-Check plugin added to all Java build pipelines) → target date → owner. This structured gap record satisfies audit evidence requirements without a commercial GRC platform.

Evidence: Retain the following for post-incident review and regulatory recordkeeping: (1) the full timeline log of when CVE-2026-33117 was publicly disclosed versus when it was detected in your environment — this gap measurement is required for NIST IR-6 (Incident Reporting) compliance and informs mean-time-to-detect (MTTD) metrics; (2) the pre-patch SBOM snapshot (captured during eradication) showing all applications carrying the vulnerable com.azure dependency coordinates, retained for a minimum period consistent with your data retention policy (NIST AU-11); (3) the lessons-learned meeting minutes and resulting CI/CD pipeline change record documenting addition of automated dependency scanning — this serves as corrective action evidence for SA-15 (Development Process, Standards, and Tools) gap closure during the next audit cycle.

Detection Guidance

Search build dependency files (pom.xml, build.gradle, gradle.lockfile) and software bills of materials for Azure SDK for Java package references (group ID: com.azure). Use your artifact registry or container scanner to confirm which deployed workloads include the affected SDK. In Azure Monitor and Entra ID sign-in logs, flag: (1) authentication events from service principals tied to SDK-dependent applications occurring outside normal operational hours or from unexpected IPs; (2) token reuse anomalies aligned with T1550 (Use Alternate Authentication Material); (3) access to Azure resources by accounts that should only operate via SDK-managed flows but are appearing via direct API calls. No public IOCs or exploitation signatures were available at analysis time; behavioral detection is the primary approach until MSRC or community researchers publish technical indicators.

Framework Mappings

MITRE-ATTACK

- **T1078** — Valid Accounts
- **T1550** — Use Alternate Authentication Material
- **T1190** — Exploit Public-Facing Application

NIST-800-53R5

- **AC-2** — Account Management
- **AC-6** — Least Privilege
- **IA-2** — Identification and Authentication (Organizational Users)
- **IA-5** — Authenticator Management
- **CA-8** — Penetration Testing

- **RA-5** — Vulnerability Monitoring and Scanning
- **SC-7** — Boundary Protection
- **SI-2** — Flaw Remediation
- **SI-7** — Software, Firmware, and Information Integrity
- **AC-3** — Access Enforcement
- **IA-8** — Identification and Authentication (Non-Organizational Users)
- **SC-13** — Cryptographic Protection

OWASP-TOP10-2021

- **A01:2021** — Broken Access Control
- **A07:2021** — Identification and Authentication Failures

CIS-V8

- **6.1** — Establish an Access Granting Process
- **6.2** — Establish an Access Revoking Process
- **6.3** — Require MFA for Externally-Exposed Applications
- **6.4** — Require MFA for Remote Network Access
- **6.5** — Require MFA for Administrative Access
- **7.3** — Perform Automated Operating System Patch Management
- **7.4** — Perform Automated Application Patch Management

SOC2-TSC

- **CC6.1** — The entity implements logical access security software, infrastructure, and architectures over protected information assets
- **CC6.3** — Authorizes, modifies, or removes access

HIPAA-SECURITY

- **164.312(a)(1)** — Access Control
- **164.312(d)** — Person or Entity Authentication

ISO-27001-2022

- **A.8.8** — Management of technical vulnerabilities
- **A.5.34** — Privacy and protection of personal information
- **A.8.24** — Use of cryptography
- **A.5.23** — Information security for use of cloud services

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1078	Valid Accounts	Defense-Evasion
T1550	Use Alternate Authentication Material	Defense-Evasion

Technique ID	Technique Name	Tactic
T1190	Exploit Public-Facing Application	Initial-Access

Sources

Source	URL	Tier
MSRC Update Guide	https://msrc.microsoft.com/update-guide/vulnerability/CVE-2026-33117	T1
(consolidated)	https://api.msrc.microsoft.com/cvrf/v3.0/cvrf/2026-May	T1
The most severe Linux threat to surface in years catches the world ...	https://arstechnica.com/security/2026/04/as-the-most-severe-linux-t...	T2
How Cloudflare responded to the “Copy Fail” Linux vulnerability	https://blog.cloudflare.com/copy-fail-linux-vulnerability-mitigation/	T3
CVE-2026-33017 Detail - NVD	https://nvd.nist.gov/vuln/detail/CVE-2026-33017	T1
NVD	https://nvd.nist.gov/vuln/detail/CVE-2026-33117	T1

Generated 2026-05-12 14:08 UTC by TJS Security Command Center