INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-11 13:36 UTC

GHSA-mf9v-mfxr-j63j: urllib3: Decompression-bomb safeguards bypassed in parts of the streaming API

CVE VULNERABILITY | MEDIUM | CVSS 5.9

SCC Item ID	SCC-CVE-2026-0154
Type	CVE Vulnerability
CVE ID	CVE-2026-44432
Severity	MEDIUM
CVSS Base Score	5.9
Affected Products	urllib3 (PyPI), specific affected versions not confirmed from available sources
Published	2026-05-11T14:51:45Z
Discovery Source	Osv

Executive Summary

A vulnerability in the urllib3 Python library allows specially crafted compressed HTTP responses to bypass memory safeguards during streaming, potentially exhausting server resources and causing service outages. Any application built with urllib3 that processes streaming HTTP responses is potentially exposed. The business risk is denial of service against internal tooling, APIs, or services that rely on this widely used library.

Technical Analysis

CVE-2026-44432 (GHSA-mf9v-mfxr-j63j) affects the urllib3 Python library. The vulnerability exists because decompression-bomb safeguards, which limit the maximum decompressed size of HTTP response bodies, are not consistently enforced across all code paths in the streaming API. An attacker controlling an HTTP endpoint (or performing a man-in-the-middle position) can serve a malicious compressed payload that decompresses to an arbitrarily large size, exhausting system memory or CPU and causing denial of service. CWE-400 (Uncontrolled Resource Consumption) and CWE-409 (Improper Handling of Highly Compressed Data) apply. MITRE ATT&CK maps to T1499.004 (Endpoint Denial of Service: Application or System Exploitation). CVSS base score: 5.9 (Medium). Specific affected version ranges were not confirmed from available sources at analysis time. Exploitation requires the application to use urllib3's streaming response handling against an attacker-controlled or compromised upstream server. No CISA KEV listing. No active threat actor exploitation reported. Source authority: GitHub advisory (GHSA-mf9v-mfxr-j63j) and NVD (CVE-2026-44432). Note: One

source references CVE-2026-4432, while the primary NVD entry is CVE-2026-44432. These are treated as the same advisory; treat CVE-2026-44432 as the canonical identifier.

Action Checklist

1. Step 1: Assessment / Identification, Identify all applications and services in your environment that import urllib3 directly or transitively (via requests, boto3, pip internals, etc.) and flag those using streaming response handling (stream=True or iter_content/iter_lines patterns). Prioritize internet-facing or third-party-data-consuming services first.
2. Step 2: Detection, Search dependency manifests (requirements.txt, Pipfile.lock, pyproject.toml, poetry.lock) and installed package inventories for urllib3. Run 'pip show urllib3' or query your software composition analysis (SCA) tool. Check application logs for anomalous memory growth or OOM events in services that make outbound HTTP requests with streaming enabled.
3. Step 3: Eradication, Monitor the urllib3 project GitHub repository and official advisory at https://github.com/advisories/GHSA-mf9v-mfxr-j63j for confirmation of the patched version. Once the fixed version is released, upgrade urllib3 in your environment. Until then, if upgrade is not immediately possible, evaluate disabling streaming response handling in affected code paths as a temporary mitigation.
4. Step 4: Recovery, After upgrading, validate that dependent libraries (requests, botocore, pip) are compatible with the new urllib3 version. Re-run application test suites. Monitor memory and CPU metrics on affected services for 24-48 hours post-deployment to confirm normal resource consumption. Verify the package version in production matches the patched release.
5. Step 5: Post-Incident, Review your software composition analysis coverage to confirm transitive dependencies are tracked. Evaluate whether urllib3 and similar foundational HTTP libraries are included in your vulnerability management scan scope. Add decompression-bomb and resource exhaustion scenarios to threat modeling for services that consume external HTTP streams.

IR / Forensic Enrichment

Triage Priority	STANDARD
Escalation Criteria	Escalate to urgent if osquery or log analysis reveals any Python service has experienced OOM kills or sustained memory growth exceeding 2x baseline while processing streaming HTTP responses from external sources, or if the environment includes regulated workloads (PII/PHI/PCI) where service disruption from a denial-of-service condition would trigger breach notification or SLA obligations.
Recovery Notes	After upgrading urllib3 to the advisory-confirmed patched version, verify the correct version is loaded at runtime — not just installed — by inspecting 'urllib3.__version__' from within running application processes where possible (e.g., via a debug endpoint or admin console), since shadowed virtualenv installations can cause the vulnerable version to remain loaded despite a successful pip upgrade. Monitor RSS memory consumption on all Python services that perform outbound streaming HTTP requests for a minimum of 48 hours post-patch, comparing against the pre-patch baselines captured during Step 1 triage. If memory profiles remain elevated or erratic after patching, treat the service as potentially impacted by a prior exploitation attempt and escalate to full heap dump analysis.

Forensic Artifacts	Linux OOM killer events in '/var/log/syslog' or 'journalctl -k' referencing Python processes — a decompression-bomb exploit of CVE-2026-44432 would exhaust heap memory in the urllib3 streaming response buffer, triggering kernel OOM kills against the consuming process before Python can raise an exception | Process memory snapshots from '/proc//smaps' and '/proc//statm' for Python processes making outbound HTTP requests with streaming enabled, showing anomalous VmRSS or VmPeak growth consistent with uncontrolled decompression buffer expansion in urllib3's response iterator | Application-layer HTTP response logs (gunicorn, uwsgi, or custom access logs) capturing outbound requests to external origins that returned Content-Encoding: gzip, br, or zstd with unexpectedly large transfer sizes, which would be the delivery mechanism for a crafted decompression-bomb payload targeting the urllib3 streaming API | Pip freeze snapshots and 'pip show urllib3' output (including 'Location:' path) from all production virtualenvs, establishing the confirmed vulnerable urllib3 version and install path across the environment at time of discovery | Contents of 'urllib3/response.py' from the deployed package directory — the specific decompression and streaming iterator logic in this file is the vulnerable code path for CVE-2026-44432, and preserving the pre-patch bytecode confirms which version was running and allows code-level verification of the vulnerability's presence

Per-Action IR Details

Step 1: Containment — Identify all applications and services in your environment that import urllib3 directly or transitively (via requests, boto3, pip internals, etc.) and flag those using streaming response handling (stream=True or iter_content/iter_lines patterns). Prioritize internet-facing or third-party-data-consuming services first.

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment Strategy: identify and bound the scope of affected systems before the threat can propagate or be further exploited

Controls: NIST IR-4 (Incident Handling), NIST SI-2 (Flaw Remediation), CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory), CIS 2.1 (Establish and Maintain a Software Inventory)

Compensating: Run 'grep -r "import urllib3\|from urllib3\|stream=True\|iter_content\|iter_lines" /opt /srv /home --include="*.py" -l' across all Python application directories to enumerate exposed code paths. Supplement with 'pip list --path | grep urllib3' on each host. For transitive exposure, run 'pipdeptree | grep -A5 urllib3' (install via pip) to map requests/boto3/pip dependency chains without a commercial SCA tool.

Evidence: Before isolating or rate-limiting any service, capture the current process memory baseline for Python services making outbound HTTP calls: run 'ps aux | grep python' and record RSS/VSZ values and PIDs. Preserve '/proc//maps' and '/proc//smaps' on Linux hosts for any Python process showing anomalous memory growth, as a decompression-bomb exploit against urllib3's streaming API would manifest as runaway heap growth within the urllib3 response buffer before an OOM kill. Also snapshot current package state with 'pip freeze > /tmp/pip_freeze_$(hostname)_$(date +%Y%m%d).txt' as a forensic baseline of the installed urllib3 version at time of triage.

Step 2: Detection — Search dependency manifests (requirements.txt, Pipfile.lock, pyproject.toml, poetry.lock) and installed package inventories for urllib3. Run 'pip show urllib3' or query your software composition analysis (SCA) tool. Check application logs for anomalous memory growth or OOM events in services that make outbound HTTP requests with streaming enabled.

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 — Detection and Analysis: correlate multiple evidence sources to confirm presence of vulnerable component and determine whether exploitation indicators exist in the environment

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

Logs)

Compensating: For manifest scanning without SCA tooling, run: 'find / -name "requirements*.txt" -o -name "Pipfile.lock" -o -name "pyproject.toml" -o -name "poetry.lock" 2>/dev/null | xargs grep -l "urllib3"' to enumerate all pinned dependency files. For OOM detection without a SIEM, query the kernel ring buffer and systemd journal: 'journalctl -k | grep -i "oom\|killed process\|out of memory"' and 'dmesg | grep -i "oom_kill\|python"'. On hosts with osquery, run 'SELECT name, version FROM python_packages WHERE name="urllib3";' across the fleet via osquery's live query interface.

Evidence: Collect Linux OOM killer logs from '/var/log/syslog' or 'journalctl -k' filtered for OOM events referencing Python processes — a successful decompression-bomb exploitation of CVE-2026-44432 would trigger kernel OOM kills against the urllib3-consuming process before a clean exception is raised. Also collect application-level logs from web frameworks (e.g., gunicorn access logs at '/var/log/gunicorn/*.log', uwsgi logs) for requests that returned abnormally large Content-Encoding: gzip or Content-Encoding: br responses to outbound requests, and capture '/proc//status' snapshots showing 'VmRSS' and 'VmPeak' growth trajectory for any flagged Python service.

Step 3: Eradication — Upgrade urllib3 to the patched version once confirmed by the urllib3 maintainers via the official advisory at https://github.com/advisories/GHSA-mf9v-mfxr-j63j. Specific fixed version was not confirmed from available sources at analysis time — verify the current patched release before upgrading. If upgrade is not immediately possible, evaluate disabling streaming response handling in affected code paths as a temporary mitigation.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication: remove the vulnerability from the environment by applying the vendor-confirmed fix, and verify eradication before returning systems to production

Controls: NIST SI-2 (Flaw Remediation), NIST CM-3 (Configuration Change Control), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: For teams without automated patch pipelines, execute the upgrade in a staging virtualenv first: 'python -m venv /tmp/urllib3_test && source /tmp/urllib3_test/bin/activate && pip install urllib3== && python -c "import urllib3; print(urllib3.__version__)"'. Then apply to production venvs with 'pip install --upgrade urllib3==' and immediately re-run 'pip show urllib3' to confirm installed version matches the advisory-specified fix. If streaming must remain enabled temporarily, add an explicit 'max_size' response guard in application code wrapping 'urllib3.response.HTTPResponse.read()' calls, or set 'URLLIB3_MAX_RESPONSE=' if the patched version exposes such a config parameter — verify in the upstream release notes before applying.

Evidence: Before upgrading, preserve a copy of the vulnerable urllib3 package for forensic reference: 'cp -r $(python -c "import urllib3; import os; print(os.path.dirname(urllib3.__file__))") /tmp/urllib3_vuln_backup_$(date +%Y%m%d)/'. This preserves the exact bytecode and source of the vulnerable decompression routines (specifically 'urllib3/response.py' and any streaming iterator methods) in case post-incident analysis requires confirming the exploit path. Also capture 'pip show urllib3' output including 'Location:' field to document which virtualenvs held the vulnerable version at time of remediation.

Step 4: Recovery — After upgrading, validate that dependent libraries (requests, botocore, pip) are compatible with the new urllib3 version. Re-run application test suites. Monitor memory and CPU metrics on affected services for 24-48 hours post-deployment to confirm normal resource consumption. Verify the package version in production matches the patched release.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery: restore systems to normal operation with verified integrity, and monitor for signs of continued exploitation or regression during the observation window

Controls: NIST SI-7 (Software, Firmware, and Information Integrity), NIST CP-10 (System Recovery and Reconstitution), NIST AU-6 (Audit Record Review, Analysis, and Reporting), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: Validate dependency compatibility without a CI/CD pipeline by running 'pip check' post-upgrade, which will surface any broken dependency constraints between the new urllib3 version and installed

requests/botocore/pip. For memory monitoring without APM tooling, deploy a simple bash watchdog: 'while true; do ps -o pid,rss,vsz,comm -p $(pgrep -d"," python) >> /tmp/mem_monitor_$(date +%Y%m%d).log; sleep 60; done &' on each affected host and review the log at 24h and 48h intervals for anomalous RSS growth. To verify production version integrity, run 'pip show urllib3 | grep Version' on each production host and compare against the expected patched version hash using 'pip hash' against a known-good wheel file.

Evidence: During the 24-48 hour observation window, continuously collect memory metrics for all Python processes that use urllib3 streaming — specifically look for services consuming data from external third parties via streaming HTTP, as these represent the realistic exploitation vector for CVE-2026-44432. Preserve time-series snapshots of '/proc//statm' for comparison against pre-patch baselines captured in Step 1. If any OOM events recur post-patch, immediately collect a core dump ('gcore ') and the corresponding 'urllib3/response.py' bytecode to determine whether the correct patched version is actually loaded at runtime versus a cached or shadowed installation.

Step 5: Post-Incident — Review your software composition analysis coverage to confirm transitive dependencies are tracked. Evaluate whether urllib3 and similar foundational HTTP libraries are included in your vulnerability management scan scope. Add decompression-bomb and resource exhaustion scenarios to threat modeling for services that consume external HTTP streams.

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity: use lessons learned to improve detection, reduce dwell time for similar vulnerabilities, and update threat models to reflect newly understood attack classes

Controls: NIST IR-4 (Incident Handling), NIST IR-8 (Incident Response Plan), NIST RA-3 (Risk Assessment), NIST SI-5 (Security Alerts, Advisories, and Directives), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 3.2 (Establish and Maintain a Data Inventory)

Compensating: For teams without a commercial SCA platform, integrate OSV-Scanner (free, Google) into CI pipelines: 'osv-scanner --lockfile requirements.txt' will flag urllib3 and transitive PyPI dependencies against the OSV vulnerability database including GHSA entries like GHSA-mf9v-mfxr-j63j. Write a Sigma rule targeting your application logs for anomalous response-size patterns: correlate HTTP response Content-Encoding: gzip/br/zstd with response sizes exceeding a configurable threshold (e.g., >50MB) in services known to use urllib3 streaming. Add a YARA rule scanning deployed Python environments for the vulnerable urllib3 version string in 'urllib3/__init__.py' as a low-cost continuous verification mechanism.

Evidence: For the lessons-learned record, compile the complete exposure inventory produced in Steps 1-2: the list of all applications with urllib3 in their dependency graph, the subset using streaming response patterns, and the subset confirmed to consume external third-party HTTP streams (the realistic attack surface for a decompression-bomb delivered via a malicious or compromised upstream server). Preserve the 'pip freeze' snapshots from all production environments captured during triage as the baseline for measuring time-to-patch SLA for foundational Python HTTP library vulnerabilities going forward.

Detection Guidance

Query your SCA tool, SBOM, or package inventory for urllib3 and compare against the official fixed version once released by the urllib3 maintainers via https://github.com/advisories/GHSA-mf9v-mfxr-j63j. For runtime detection, monitor application memory usage and CPU spikes in services that make outbound HTTP requests with streaming enabled; abnormal resource exhaustion on response receipt is the primary behavioral indicator. In Python application logs, look for MemoryError exceptions or OOM killer events (Linux: /var/log/syslog or journalctl for 'Out of memory' entries) correlated with HTTP client activity. No network-layer IOCs are available for this vulnerability; detection is host-based and dependency-inventory-based. EPSS score is 0.0 at time of analysis, indicating low observed exploitation probability.

Framework Mappings

MITRE-ATTACK

- **T1499.004** — Application or System Exploitation

NIST-800-53R5

- **SC-5** — Denial-of-Service Protection

CIS-V8

- **13.8** — Deploy a Network Intrusion Prevention Solution

ISO-27001-2022

- **A.8.8** — Management of technical vulnerabilities

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1499.004	Application or System Exploitation	Impact

Sources

Source	URL	Tier
osv	https://osv.dev/vulnerability/GHSA-mf9v-mfxr-j63j	T3
CVE-2026-4432 Detail - NVD	https://nvd.nist.gov/vuln/detail/CVE-2026-4432	T1
urllib3: Decompression-bomb safeguards bypassed in parts of the ...	https://github.com/advisories/GHSA-mf9v-mfxr-j63j	T3
SB20260509126 - Multiple vulnerabilities in urllib3	https://www.cybersecurity-help.cz/vdb/SB20260509126	T3
Dirty Frag, a new Linux Local Privilege Escalation vulnerability, was ...	https://ccb.belgium.be/advisories/warning-dirty-frag-new-linux-loca...	T3
NVD	https://nvd.nist.gov/vuln/detail/CVE-2026-44432	T1

Generated 2026-05-11 13:36 UTC by TJS Security Command Center