INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-07-06 06:34 UTC

Trail of Bits fickling MLAllowlist Security Check Bypass via UnsafeImportsML Analysis Pass (CVE-2026-14535)

CVE VULNERABILITY | MEDIUM | CVSS 5.3

SCC Item ID	SCC-CVE-2026-0388
Type	CVE Vulnerability
CVE ID	CVE-2026-14535
Severity	MEDIUM
CVSS Base Score	5.3
EPSS Score	0.0030 (22th percentile)
Affected Products	Trail of Bits fickling versions up to and including 0.1.11
Published	2026-07-04
Discovery Source	Gemini

Executive Summary

A logic flaw in Trail of Bits fickling, a Python library used to inspect and gate untrusted pickle files, allows malicious pickle files to bypass the tool's ML-based import allowlist entirely. Any pipeline using fickling versions 0.1.11 or earlier as a security gate for pickle deserialization may be accepting files it should block, without generating any alert. Organizations using fickling to screen model files or serialized data in AI/ML pipelines should treat this as a gap in a defensive control, not a direct system compromise.

Technical Analysis

CVE-2026-14535 (CWE-693, Protection Mechanism Failure) affects Trail of Bits fickling versions up to and including 0.1.11. The flaw exists in how two analysis passes share state: UnsafeImportsML unconditionally calls AnalysisContext.shorten_code(node) on every import node it processes, which registers each node in a shared set and sets already_reported=True. The MLAllowlist analysis pass then evaluates the same nodes, reads already_reported=True, and interprets each import as already processed, skipping its security checks entirely. The practical result is that pickle files containing imports that MLAllowlist would normally block can pass fickling analysis without flagging. MITRE techniques T1059.006 (Command and Scripting Interpreter: Python) and T1195.001 (Supply Chain Compromise: Compromise Software Dependencies and Development Tools) are relevant given fickling's role as a pipeline gate. CVSS base score is reported at 5.3 (medium). EPSS score is 0.00304 (22nd percentile), indicating low current exploitation probability. No CISA KEV listing and no confirmed

active exploitation are present in the provided source data. The vulnerability is reported across multiple sources including NVD and third-party security outlets; NVD detail page exists but full CVSS vector and scoring data should be confirmed at the NVD record directly.

Action Checklist

1. Step 1: Identify & Isolate. Identify all pipeline stages, CI/CD jobs, and automated workflows that invoke fickling for pickle file analysis. Temporarily treat any pickle files processed by fickling 0.1.11 or earlier as unvetted and hold them for re-analysis until a fixed version is applied. Do not rely on recent fickling pass results as confirmation of file safety.
2. Step 2: Detection. Audit fickling invocations in your codebase and pipeline logs. Query your code repositories and dependency manifests (requirements.txt, pyproject.toml, setup.cfg, Pipfile.lock) for fickling<=0.1.11. Check pipeline logs for any MLAllowlist analysis runs; any pass result from an affected version may be a false negative. No external network IOCs are present in the provided source data for this vulnerability.
3. Step 3: Eradication. Upgrade fickling to the patched release addressing CVE-2026-14535. Confirm the installed version post-upgrade. Re-run MLAllowlist analysis against any pickle files that were processed and passed by the vulnerable version. Reference the Trail of Bits fickling GitHub repository (github.com/trailofbits/fickling) releases page for the first version tagged after 0.1.11; upgrade to that version or later. Consult official release notes for the patched version number.
4. Step 4: Recovery. After upgrading, validate that MLAllowlist analysis is producing expected results by running fickling against known-malicious pickle test files. Monitor pipeline outputs for newly flagged files that previously passed; this would indicate the fix is working and prior false negatives existed. Re-enable automated pickle gating only after confirming correct behavior. NIST AU-6 (Audit Record Review, Analysis, and Reporting) applies: review analysis logs from the vulnerable period for any anomalous files that passed without inspection.
5. Step 5: Post-Incident. Evaluate whether fickling was the sole control for pickle deserialization safety or one of several layers. If it was the only gate, implement defense-in-depth: add secondary static analysis, restrict pickle deserialization to explicitly trusted sources (NIST AC-3, Access Enforcement; NIST AC-4, Information Flow Enforcement), and apply CIS 7.1 (Establish and Maintain a Vulnerability Management Process) to ensure developer security libraries are included in routine dependency scanning. Document this gap in your risk register.

IR / Forensic Enrichment

Triage Priority	URGENT
Escalation Criteria	Escalate to incident commander and legal/compliance if re-analysis of pickle files processed during the vulnerable window (fickling <=0.1.11) reveals any file now flagged as malicious by the patched version, as this indicates potentially untrusted code was deserialized in production AI/ML pipelines and may constitute a data integrity or supply chain compromise requiring breach notification assessment.

Recovery Notes	Re-enable automated pickle gating only after the patched fickling version has been confirmed active in all pipeline environments and has correctly flagged at least one known-malicious test pickle file during validation. Monitor pipeline outputs for a minimum of 30 days post-upgrade for newly flagged files that previously passed, treating any such finding as confirmation of a prior false negative requiring immediate retroactive investigation of the model or data artifact and its downstream consumers. Verify that fickling and all other security-function dependencies are enrolled in automated CVE scanning (pip-audit or equivalent) so a future regression in a security library dependency is caught at ingestion rather than post-exploitation.
Forensic Artifacts	CI/CD pipeline execution logs showing fickling MLAllowlist invocations: capture log lines containing the fickling command, arguments (pickle file path), stdout verdict ('passed' or equivalent), return code, and execution timestamp for every run during the vulnerable window — these are the primary record of what the broken gate allowed through. | Dependency manifest snapshots (requirements.txt, pyproject.toml, Pipfile.lock, setup.cfg) from all affected repositories and container image build contexts, preserving the pinned or constrained fickling version string that confirms the vulnerable version was active at the time of each pipeline run. | SHA-256 hash inventory of all pickle files (.pkl, .joblib, or pickle-serialized .pt/.pth model files) that received a fickling MLAllowlist pass result during the vulnerable period — required to match files for retroactive re-analysis and to establish whether any file now flagged by the patched version was previously admitted into the pipeline. | fickling analysis output records (stdout/stderr) from both the original vulnerable-version runs and the post-upgrade re-analysis runs, preserved side-by-side per file hash, to document the delta between what the broken MLAllowlist gate passed and what the patched version detects — this is the core forensic evidence of false-negative scope. | Container image layer history and virtual environment pip freeze outputs from all pipeline runner environments, capturing the exact fickling version installed at build time and providing a reproducible baseline for confirming eradication of the vulnerable version across all execution contexts.

Per-Action IR Details

Step 1: Containment — Identify all pipeline stages, CI/CD jobs, and automated workflows that invoke fickling for pickle file analysis. Temporarily treat any pickle files processed by fickling 0.1.11 or earlier as unvetted and hold them for re-analysis until a fixed version is applied. Do not rely on recent fickling pass results as confirmation of file safety.

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment Strategy: stop the bleeding by isolating the affected control from the pipeline trust chain; fickling pass results from vulnerable versions are untrusted attestations and must be treated as void.

Controls: NIST AC-4 (Information Flow Enforcement), CIS 4.6 (Securely Manage Enterprise Assets and Software)

Compensating: Run `pip show fickling` or `grep -r 'fickling' requirements.txt pyproject.toml Pipfile.lock setup.cfg` across all repos to enumerate affected pipeline nodes. For each CI/CD job invoking fickling, add a temporary pipeline gate (e.g., a pre-step bash check: `python -c "import fickling; print(fickling.__version__)" && exit 1` to force-fail jobs on vulnerable versions) until upgrade is confirmed.

Evidence: Before halting or quarantining any pickle files already in flight, snapshot the pipeline execution state: capture CI/CD job logs showing fickling invocation arguments, the list of pickle files passed to fickling with their SHA-256 hashes (run `sha256sum *.pkl` or equivalent), and the MLAllowlist analysis output records (stdout/stderr from fickling runs) for all files processed since the last known-good version. These logs are the only record of what was allowed through the broken gate and will be needed for retroactive triage.

Step 2: Detection — Audit fickling invocations in your codebase and pipeline logs. Query your code repositories and dependency manifests (requirements.txt, pyproject.toml, setup.cfg, Pipfile.lock) for fickling<=0.1.11. Check pipeline logs for any MLAllowlist analysis runs; any pass result from an affected version may be a false negative. No external network IOCs are present in the provided source data for this vulnerability.

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 — Detection and Analysis: correlate dependency manifests and pipeline execution logs to establish the scope of files that transited the broken fickling MLAllowlist gate, treating every pass result as a potential false negative requiring re-analysis.

Controls: NIST AU-6 (Audit Record Review, Analysis, and Reporting), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 2.1 (Establish and Maintain a Software Inventory)

Compensating: Use `grep -rn 'MLAllowlist\|fickling' ./ --include='*.py' --include='*.yml' --include='*.yaml'` to locate all invocation points in source and CI config files. For dependency auditing, run `pip-audit` (free, PyPA tool) or `safety check` against the project's requirements files to flag fickling<=0.1.11. Export CI/CD job logs (GitHub Actions: `gh run list` + `gh run view --log`; GitLab CI: pipeline job trace API) and grep for fickling stdout lines containing 'Analysis passed' or 'No unsafe' to enumerate all false-negative candidates.

Evidence: The primary artifacts for this detection step are: (1) dependency manifest files (requirements.txt, pyproject.toml, Pipfile.lock, setup.cfg) showing the pinned or constrained fickling version; (2) CI/CD pipeline execution logs — specifically any log line where fickling was invoked with the MLAllowlist pass, including the pickle file path, invocation timestamp, and return code; (3) the list of pickle/model files (e.g., .pkl, .pt serialized via pickle, .joblib) that received a fickling pass result during the vulnerable window. Preserve these before any upgrade action, as post-upgrade re-runs will not reconstruct what the broken gate allowed through.

Step 3: Eradication — Upgrade fickling to the patched release addressing CVE-2026-14535. Confirm the installed version post-upgrade. Re-run MLAllowlist analysis against any pickle files that were processed and passed by the vulnerable version. Reference the Trail of Bits fickling GitHub repository and official release notes for the patched version number; the provided source data does not specify a confirmed fixed release tag.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication: remove the vulnerable fickling version from all pipeline environments and verify eradication by confirming the patched version is active and MLAllowlist analysis produces correct results on known-malicious test payloads.

Controls: NIST SI-2 (Flaw Remediation), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management)

Compensating: Perform the upgrade via `pip install --upgrade fickling` and immediately verify with `python -c "import fickling; print(fickling.__version__)"` in every virtual environment, container image, and CI runner base image where fickling is installed. For container-based pipelines, rebuild affected Docker images from scratch with the pinned patched version and push to the registry — do not rely on layer caching. After upgrade, re-run `fickling --check-safety .pkl` (or the equivalent MLAllowlist invocation) against the full set of pickle files identified in Step 2 as having previously passed the broken gate.

Evidence: Before executing the pip upgrade or container rebuild (actions that alter the live environment state), capture: (1) the exact installed fickling version string from every affected environment (`pip show fickling` output saved to a text file per environment); (2) a frozen copy of the full dependency tree (`pip freeze > pre-upgrade-freeze.txt`) to establish the pre-eradication baseline for later comparison; (3) SHA-256 hashes of all pickle files queued for re-analysis, so post-upgrade results can be definitively matched to the same files that transited the broken gate. These artifacts support both the re-analysis workflow and any post-incident audit of what was processed under the vulnerable version.

Step 4: Recovery — After upgrading, validate that MLAllowlist analysis is producing expected results by running fickling against known-malicious pickle test files. Monitor pipeline outputs for newly flagged files that

previously passed — this would indicate the fix is working and prior false negatives existed. Re-enable automated pickle gating only after confirming correct behavior. NIST AU-6 (Audit Record Review, Analysis, and Reporting) applies: review analysis logs from the vulnerable period for any anomalous files that passed without inspection.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery: restore the pickle gating pipeline to trusted operation only after verifying the patched fickling version correctly detects malicious imports on known-bad test cases, and monitor for newly flagged files as evidence of prior false negatives during the vulnerable window.

Controls: NIST AU-6 (Audit Record Review, Analysis, and Reporting), CIS 8.2 (Collect Audit Logs), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: Source known-malicious pickle test files from the fickling project's own test suite on GitHub (the repository includes crafted malicious pickle payloads used in unit tests) and run the patched fickling MLAllowlist against them to confirm detection. For pipeline monitoring without a SIEM, add a post-fickling step that writes all analysis results (file hash, verdict, timestamp, fickling version) to an append-only log file and configure a cron job or CI post-step to alert (email or Slack webhook) on any 'previously-passed / now-flagged' delta by diffing current results against the Step 2 captured pass list.

Evidence: The volatile/forensic evidence to preserve before re-enabling the pipeline gate is the full set of fickling analysis output records from the re-analysis runs performed in Step 3 — specifically the per-file verdict (pass/fail), the fickling version string confirming the patched version was active, and the SHA-256 hash of each analyzed file. This creates an auditable record proving which files were cleared by the fixed gate versus which were flagged as previously missed. Retain pipeline execution logs showing the re-enable timestamp and the operator who authorized the gate restoration.

Step 5: Post-Incident — Evaluate whether fickling was the sole control for pickle deserialization safety or one of several layers. If it was the only gate, implement defense-in-depth: add secondary static analysis, restrict pickle deserialization to explicitly trusted sources (NIST AC-3, Access Enforcement; NIST AC-4, Information Flow Enforcement), and apply CIS 7.1 (Establish and Maintain a Vulnerability Management Process) to ensure developer security libraries are included in routine dependency scanning. Document this gap in your risk register.

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity: conduct lessons-learned review focused on the single-point-of-failure architecture where fickling was the sole pickle deserialization gate, update the risk register to reflect the dependency on a security library that itself was not subject to routine vulnerability scanning, and improve detection coverage so future regressions in security tooling dependencies are caught before exploitation.

Controls: NIST AC-3 (Access Enforcement), NIST AC-4 (Information Flow Enforcement), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 2.3 (Address Unauthorized Software)

Compensating: Add fickling (and all other security-function Python dependencies) to a `pip-audit` or `safety` scan step in every CI/CD pipeline, configured to fail the build on any known CVE in a security library. As a secondary static analysis layer, integrate `picklescan` (free, open-source) alongside fickling so no single tool's logic flaw creates a blind spot. For access restriction, enforce that pickle deserialization only occurs within an isolated subprocess or container with no network access and read-only filesystem mounts for all paths outside the designated model directory (achievable with Docker `--read-only` and `--network none` flags).

Evidence: For the post-incident record, compile: (1) the complete list of pickle files that transited the vulnerable fickling gate during the exposure window, with their SHA-256 hashes and pipeline source provenance; (2) the results of re-analysis under the patched version, indicating how many files were retrospectively flagged as malicious (if any); (3) the dependency manifest snapshots from Step 3 showing the pre- and post-upgrade fickling version; and (4) the risk register entry documenting the control gap — specifically that fickling was not included in routine CVE scanning for developer security dependencies. These artifacts form the evidentiary basis for any regulatory disclosure assessment and the lessons-learned report.

Detection Guidance

Detection centers on dependency inventory rather than behavioral or network IOCs, as no active exploitation indicators are present in the provided source data. Query all package manifests and lock files across repositories and build environments for fickling versions 0.1.11 and earlier. In Python environments, run 'pip show fickling' or equivalent to confirm installed version. Search CI/CD pipeline definitions (GitHub Actions workflows, Jenkins pipelines, GitLab CI YAML) for fickling invocations. Review any pipeline stage that calls fickling's MLAllowlist pass and treat all pass results from the vulnerable version as potentially unreliable. If your environment logs Python package invocations, query for fickling execution events during the exposure window. No network-layer IOCs (IPs, domains, hashes) are present in the provided source data for this CVE. NIST AU-2 (Event Logging) and CIS 8.2 (Collect Audit Logs) apply as foundational controls for maintaining the pipeline visibility needed to perform this review.

Framework Mappings

MITRE-ATTACK

- **T1059.006** — Python
- **T1195.001** — Compromise Software Dependencies and Development Tools

ISO-27001-2022

- **A.8.8** — Management of technical vulnerabilities
- **A.5.21** — Managing information security in the ICT supply chain

CIS-V8

- **14.2** — Train Workforce Members to Recognize Social Engineering Attacks
- **15.1** — Establish and Maintain an Inventory of Service Providers
- **8.2** — Collect Audit Logs

NIST-800-53R5

- **AT-2** — Literacy Training and Awareness
- **SR-2** — Supply Chain Risk Management Plan
- **SI-4** — System Monitoring
- **IR-5** — Incident Monitoring

NIST-CSF-2

- **GV.SC-01** — Cybersecurity supply chain risk management program
- **DE.CM-01** — Networks and network services are monitored
- **DE.AE-08** — Incidents are declared when adverse events meet the defined incident criteria

SOC2-TSC

- **CC9.2** — Manages risks associated with vendors and business partners

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1059.006	Python	Execution
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access

Sources

Source	URL	Tier
gemini	https://www.tenable.com/cve/2026-14535	T1
CVE-2026-14535 Detail - NVD	https://nvd.nist.gov/vuln/detail/CVE-2026-14535	T1
CVE-2026-14535 in fickling	https://vuldb.com/cve/CVE-2026-14535	T3
CVE-2026-14535: CWE-693 Protection Mechanism Failure in ...	https://radar.offseq.com/threat/cve-2026-14535-cwe-693-protection-m...	T3
Trail of Bits fickling Deserialization Bypass (CVE-2026-14535)	https://www.thehackerwire.com/trail-of-bits-fickling-deserializatio...	T3

Generated 2026-07-06 06:34 UTC by TJS Security Command Center