INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-03 06:18 UTC

CVE-2025-13030: All versions of the package django-mdeditor are vulnerable to Missing Authentication for Critical Fu...

CVE VULNERABILITY | HIGH | CVSS 7.1

SCC Item ID	SCC-CVE-2026-0114
Type	CVE Vulnerability
CVE ID	CVE-2025-13030
Severity	HIGH
CVSS Base Score	7.1
EPSS Score	0.0006 (17th percentile)
Affected Products	django-mdeditor (all versions)
Published	2026-04-30T06:16:14.860
Discovery Source	Nvd

Executive Summary

CVE-2025-13030 is a high-severity vulnerability in django-mdeditor, a Django-based rich text editing package, that allows unauthenticated attackers to upload malicious files and execute arbitrary code on the host server. All versions of the package are affected, and no authentication or file sanitization controls exist on the vulnerable endpoint. Any web application built on Django that uses this package for content editing is at risk of full server compromise.

Technical Analysis

CVE-2025-13030 affects all versions of the django-mdeditor Python package (PyPI). The image upload endpoint exposed by the package lacks authentication controls (CWE-306) and does not sanitize uploaded file names, permitting unauthenticated HTTP POST requests containing malicious files. Successful exploitation can achieve arbitrary code execution (ACE) on the underlying host. CVSS base score: 7.1 (High). EPSS: 0.056% (17th percentile, low observed exploitation activity at time of publication). Not listed on CISA KEV. Relevant MITRE ATT&CK techniques: T1190 (Exploit Public-Facing Application), T1105 (Ingress Tool Transfer), T1059 (Command and Scripting Interpreter). No vendor-assigned CVSS vector was provided at time of publication. No patch version has been confirmed in the sourced data; the package appears to be unmaintained or pending remediation. Source: NVD (https://nvd.nist.gov/vuln/detail/CVE-2025-13030).

Action Checklist

1. 1. Inventory: Identify all Django applications in your environment that import or depend on django-mdeditor (check requirements.txt, Pipfile, pyproject.toml). Document affected systems and owners.
2. 2. Containment (Immediate): Restrict network access to the image upload endpoint at the WAF or reverse proxy layer. Block all unauthenticated POST requests to mdeditor upload paths (commonly /mdeditor/uploads/ or similar). If internet-facing, take the affected application offline until remediation is complete.
3. 3. Detection: Review web server and application logs for unauthenticated POST requests to mdeditor upload endpoints. Look for HTTP 200/201 responses. Inspect the upload directory for unexpected executable file types (.php, .py, .sh, .jsp, .aspx). Check for anomalous processes spawned by the web application user account.
4. 4. Eradication: Remove or disable django-mdeditor if no patched version is available. If a hard dependency, implement authentication middleware on the upload endpoint as an interim control and enforce allowed MIME types and file extensions server-side. Monitor PyPI for patched releases.
5. 5. Recovery: Audit the upload directory and application file system for suspicious files uploaded during exposure window. Hash, quarantine, and remove unauthorized files. Rotate credentials for service accounts and database connections. Validate the endpoint is no longer accessible without authentication.
6. 6. Post-Incident: Document the gap and update software composition analysis (SCA) tooling to flag django-mdeditor and similar packages. Implement policy requiring authentication review for all third-party Django packages with file upload functionality. Add unauthenticated file upload detection to WAF ruleset.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to senior IR leadership and initiate breach notification review if any file in the django-mdeditor upload directory has an executable extension (.php, .py, .sh, .jsp), if anomalous child processes are confirmed under the web application service account, or if the affected application handles PII, PHI, or payment data — any of these conditions indicates active exploitation and potential data exposure triggering regulatory notification obligations.
Recovery Notes	After eradication, monitor the application's web server access logs and Django application logs continuously for 72 hours for any resumed POST attempts to '/mdeditor/uploads/' or successor paths, as attackers who achieved initial access may return to previously identified endpoints. Verify the clean-state file system baseline (SHA-256 manifest) against the production upload directory at 24 and 72 hours post-recovery to detect any re-introduction of malicious files. If a webshell was confirmed, treat the entire application server as potentially compromised and consider full OS reimaging rather than in-place recovery, per NIST 800-61r3 §3.5 guidance on systems with evidence of attacker persistence.

Forensic Artifacts	Web server access logs (nginx /var/log/nginx/access.log or Apache /var/log/apache2/access.log): filter for POST requests to '/mdeditor/uploads/' returning HTTP 200 without a sessionid cookie — this is the direct fingerprint of CVE-2025-13030 exploitation. | django-mdeditor MEDIA_ROOT upload directory (path defined by MDE_UPLOAD_PATH in Django settings.py): any file with an executable extension (.php, .py, .sh, .jsp, .aspx) or a file whose true MIME type (per libmagic) does not match its extension is evidence of a malicious upload via the unauthenticated endpoint. | OS process accounting or auditd logs: entries showing processes spawned with parent PID matching the gunicorn, uwsgi, or mod_wsgi worker process and running as the web application user (e.g., www-data) that are not Python interpreter instances — indicative of a webshell executing OS commands post-upload. | Django application log (target defined in settings.py LOGGING): any FileResponse or view execution log entries for the mdeditor upload view that lack an associated authenticated user — Django logs the view name and, if configured, the session/user context, making this a secondary corroboration source. | Network connection state at time of discovery (/proc/net/tcp, output of 'ss -tulnp'): outbound connections from the web application server process to unexpected external IPs on non-standard ports are evidence of a reverse shell established via an uploaded webshell exploiting CVE-2025-13030.

Per-Action IR Details

Containment — Identify all Django applications in your environment that import or depend on django-mdeditor (check requirements.txt, Pipfile, pyproject.toml). If found, immediately restrict network access to the image upload endpoint at the WAF or reverse proxy layer. Block unauthenticated POST requests to any URL path associated with mdeditor upload views (commonly '/mdeditor/uploads/' or similar). If internet-facing, take the affected application offline until remediation is complete.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST SI-4 (System Monitoring), CIS 4.4 (Implement and Manage a Firewall on Servers), CIS 2.1 (Establish and Maintain a Software Inventory)

Compensating: Run 'grep -r django-mdeditor /var/www/ --include=requirements.txt --include=Pipfile --include=pyproject.toml' across all application directories to enumerate affected apps. At the nginx or Apache layer, add a location block or RewriteRule to return 403 on POST to '/mdeditor/uploads/' immediately — no SIEM required. Example nginx rule: 'location ~* /mdeditor/uploads/ { limit_except GET { deny all; } }'. For iptables: 'iptables -I INPUT -p tcp --dport 80 -m string --string "/mdeditor/uploads/" --algo bm -j DROP'.

Evidence: Before restricting the endpoint, capture a full snapshot of the django-mdeditor upload directory (commonly MEDIA_ROOT/uploads/ or as configured in settings.py MDE_UPLOAD_PATH). Run 'ls -la --full-time' and 'find . -newer /var/log/app/deploy.log -type f' to identify files created after the application went live. Preserve the web server access log (e.g., /var/log/nginx/access.log or /var/log/apache2/access.log) with a timestamped copy before any log rotation occurs. Capture the current process list ('ps aux') and active network connections ('ss -tulnp' or 'netstat -anp') from the affected server to detect any already-running reverse shells spawned by the web application service user (e.g., www-data, gunicorn).

Detection — Review web server and application logs for unauthenticated POST requests to mdeditor upload endpoint paths. Look for HTTP 200 responses to POST requests from unauthenticated sessions. Inspect the upload directory configured for django-mdeditor for unexpected file types (.php, .py, .sh, .jsp, .aspx, or other executable extensions). Check for new processes spawned by the web application user account that are not consistent with normal operation.

NIST Phase: Detection Analysis

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

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

Compensating: Parse web server access logs with: 'grep -E "POST /mdeditor/uploads/" /var/log/nginx/access.log | grep " 200 "' to isolate successful unauthenticated uploads. To identify sessions without an authenticated session cookie, correlate POST requests lacking a 'sessionid' cookie header: 'grep "POST /mdeditor/uploads/" access.log | grep -v "sessionid="'. For uploaded webshells, use 'find MEDIA_ROOT -type f \(-name "*.php" -o -name "*.py" -o -name "*.sh" -o -name "*.jsp" \) -newer /path/to/requirements.txt' to find files dropped after deployment. Deploy a YARA rule scanning the upload directory for common webshell patterns: 'yara -r webshell.yar /var/www/media/uploads/'. Check for anomalous child processes of gunicorn/uwsgi with: 'ps --ppid $(pgrep gunicorn) -o pid,comm,args'.

Evidence: Preserve raw web server access logs (nginx: /var/log/nginx/access.log; Apache: /var/log/apache2/access.log; gunicorn: check systemd journal via 'journalctl -u gunicorn --since "2025-01-01"') covering the full exposure window since django-mdeditor was first deployed. Capture Django application logs (settings.py LOGGING configuration target) for any FileResponse or upload-related view exceptions. Record the inode creation timestamps of all files in the MEDIA_ROOT upload subdirectory using 'stat -c "%n %y %z" *' — inode change time (ctime) is harder for an attacker to spoof than mtime. Capture /proc/ entries for any suspicious PIDs spawned under the web server user: 'ls -la /proc/[pid]/exe' to confirm binary paths.

Eradication — Remove or disable the django-mdeditor package if no patched version is available. If the package is a hard dependency, implement authentication middleware on the upload endpoint as an interim control and restrict allowed MIME types and file extensions server-side. Monitor PyPI and the package repository for a patched release. If a patched version is released, upgrade immediately and verify the upload endpoint enforces authentication post-upgrade.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST SI-2 (Flaw Remediation), NIST CM-7 (Least Functionality) — implied by restricting MIME types and disabling unused upload capability, CIS 7.4 (Perform Automated Application Patch Management), CIS 2.2 (Ensure Authorized Software is Currently Supported)

Compensating: If removal is not immediately possible, add a Django middleware class that enforces request.user.is_authenticated on any path matching '/mdeditor/uploads/' and returns HttpResponseForbidden(). Add this to MIDDLEWARE in settings.py above the mdeditor URL conf. For MIME restriction without enterprise tooling, use python-magic in a custom upload handler to validate true file type against a whitelist (image/jpeg, image/png, image/gif only) regardless of the Content-Type header supplied by the client. Monitor PyPI for a patched release using: 'pip index versions django-mdeditor' run daily via cron and alert on version change. Pin the package to a non-functional stub in requirements.txt ('django-mdeditor==0.0.0') to block accidental re-installation until a verified patch is available.

Evidence: Before package removal, preserve the installed package state: 'pip show django-mdeditor' (version, location, dependencies) and copy the full package directory from site-packages for offline forensic analysis of the vulnerable upload view code. Document the exact URL conf registered by the package ('grep -r mdeditor /path/to/project/urls.py') to confirm all upload routes are identified. If any webshells were uploaded, hash them with SHA-256 ('sha256sum [file]') and preserve copies in a quarantine directory with restricted permissions before deletion — these are forensic evidence and may be needed for law enforcement or insurance.

Recovery — After removing or mitigating the package, audit the upload directory and application file system for any files uploaded during the exposure window. Hash and quarantine suspicious files before deletion. Rotate credentials for any service accounts or database connections accessible from the affected application server. Confirm the endpoint is no longer accessible without authentication using an external scanner or manual test before returning the application to production.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST IA-5 (Authenticator Management) — credential rotation, NIST AU-11 (Audit Record Retention), CIS 5.2 (Use Unique Passwords), CIS 6.2 (Establish an Access Revoking Process)

Compensating: Validate endpoint closure without enterprise tooling using curl from an external host (not the server itself): 'curl -X POST https://[target]/mdeditor/uploads/ -F "file=@test.txt" -v' — confirm response is 401, 403, or connection refused. For upload directory audit, use 'find MEDIA_ROOT -type f -printf "%T+ %p\n" | sort' to list all files by creation time and cross-reference against legitimate content management activity logs. For credential rotation, enumerate all secrets in the Django settings.py (DATABASE_URL, SECRET_KEY, third-party API keys, SMTP credentials, cloud storage keys such as AWS_ACCESS_KEY_ID) and rotate each — a compromised server means all in-process secrets should be treated as exposed. Use 'grep -r "PASSWORD\|SECRET\|KEY\|TOKEN" /path/to/settings.py' to enumerate targets for rotation.

Evidence: Capture a final state snapshot of the upload directory after quarantine and before returning to production: 'find MEDIA_ROOT -type f -exec sha256sum {} \;' to establish a clean-state baseline. Preserve all rotated credential metadata (which credentials, rotation timestamp, issuing system) as part of the incident record per NIST AU-11 (Audit Record Retention). Document the external validation test result (curl output, timestamp, source IP) as evidence that the endpoint is no longer exploitable — this record supports post-incident reporting and any regulatory notification requirements.

Post-Incident — Document the gap: developer dependencies were not screened for unauthenticated file upload endpoints before deployment. Add django-mdeditor (and similar editor packages) to your software composition analysis (SCA) tooling blocklist until a patched version is confirmed. Implement a policy requiring authentication review of all third-party Django packages that expose file upload functionality. Add unauthenticated file upload detection to your WAF ruleset as a standing control.

NIST Phase: Post Incident

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

Controls: NIST IR-4 (Incident Handling), NIST IR-8 (Incident Response Plan), NIST SI-2 (Flaw Remediation), NIST SA-11 (Developer Testing and Evaluation) — requiring pre-deployment security review of third-party packages, CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 2.3 (Address Unauthorized Software)

Compensating: Implement free SCA scanning in the CI pipeline using pip-audit ('pip-audit -r requirements.txt') or Safety ('safety check -r requirements.txt') to flag known-vulnerable packages at build time. For the WAF blocklist, add a ModSecurity or NGINX rule matching POST requests to any path containing 'upload' or 'mdeditor' without a valid session cookie — export this as a Sigma rule for teams using log-based detection. Create a pre-deployment checklist item: for any Django package that registers URL patterns, manually review the urls.py for file upload views and verify each requires @login_required or equivalent. Document CVE-2025-13030 in your lessons-learned register with the root cause: no pre-deployment SCA scan and no authentication control review for third-party package URL registrations.

Evidence: Retain the full incident timeline log (first exposure date based on package install timestamp, first detected exploit attempt from logs, containment timestamp, eradication timestamp) as required by NIST IR-5 (Incident Monitoring) for tracking and post-incident review. Preserve the original requirements.txt or Pipfile that included the vulnerable django-mdeditor version as a reference artifact for the process gap documentation. If any webshells or suspicious files were recovered, retain their SHA-256 hashes and a sanitized copy for IOC sharing with sector peers or CISA if exploitation was confirmed.

Detection Guidance

Query web access logs for POST requests to paths containing 'mdeditor' or 'uploads' where no session cookie or Authorization header is present and the HTTP response code is 200 or 201. In Django application logs, look for file write events in the media or upload directory from unauthenticated request contexts. If file integrity monitoring (FIM) is deployed on the application server, alert on new executable-extension files appearing in web-accessible directories. For host-based detection, monitor for child processes spawned by the WSGI/ASGI worker process (e.g., gunicorn, uWSGI) that execute shells or interpreters (bash, python, sh). MITRE T1190 and

T1105 detection coverage in your SIEM should surface ingress tool transfer activity if a payload was staged post-upload.

Framework Mappings

MITRE-ATTACK

- **T1059** — Command and Scripting Interpreter
- **T1190** — Exploit Public-Facing Application
- **T1105** — Ingress Tool Transfer

NIST-800-53R5

- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **SI-7** — Software, Firmware, and Information Integrity
- **CA-8** — Penetration Testing
- **RA-5** — Vulnerability Monitoring and Scanning
- **SC-7** — Boundary Protection
- **SI-2** — Flaw Remediation
- **CA-7** — Continuous Monitoring
- **IA-2** — Identification and Authentication (Organizational Users)

OWASP-TOP10-2021

- **A07:2021** — Identification and Authentication Failures

CIS-V8

- **6.3** — Require MFA for Externally-Exposed Applications

ISO-27001-2022

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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1059	Command and Scripting Interpreter	Execution
T1190	Exploit Public-Facing Application	Initial-Access
T1105	Ingress Tool Transfer	Command-And-Control

Sources

Source	URL	Tier
nvd	https://nvd.nist.gov/vuln/detail/CVE-2025-13030	T1
CVE-2025-13030 | Tenable®	https://www.tenable.com/cve/CVE-2025-13030	T3
CVE-2025-13030 — Missing Authentication in Django-Mdeditor	https://dbugs.ptsecurity.com/vulnerability/PT-2026-36039	T3
CVE-2025-13030 - High Vulnerability - TheHackerWire	https://www.thehackerwire.com/vulnerability/CVE-2025-13030/	T3
CVE-2025-13030 - Security Bug Tracker - Debian	https://security-tracker.debian.org/tracker/CVE-2025-13030	T3