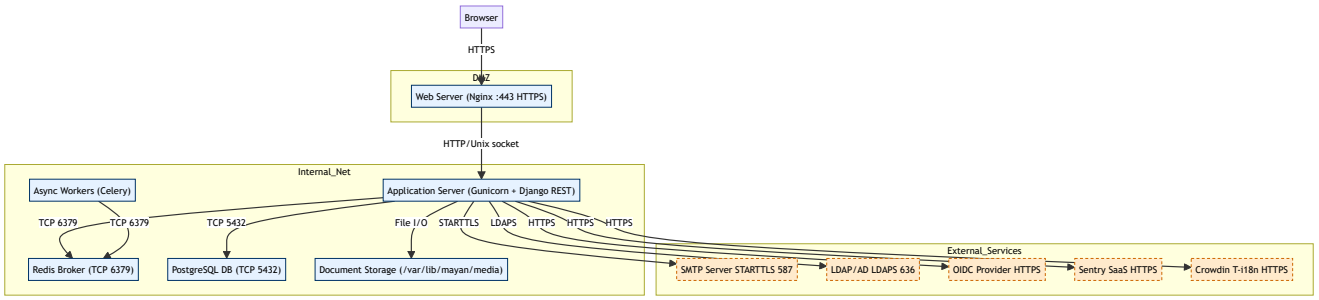
Mayan-EDMS Security Architecture Report

Repository analysed: https://github.com/mayan-edms/Mayan-EDMS (Last pull: June 2025)

Assumptions & Scope

- Assessment is limited to publicly available source code, Dockerfiles and online documentation (v4.6.x branch).
- No proprietary plugins or deployment-specific hardening options were provided.
- Code review was static (no dynamic testing). External network topology was inferred from Docker compose examples.
- Where a version pin was absent, the latest release at time of report is assumed.

Logical System Architecture



---

1. Authentication

Description

- Mechanisms: Django built-in username/password backend; optional email login. Addon apps enable LDAP (`django-auth-ldap`) and OIDC (`mozilla-django-oidc`). TOTP 2-factor introduced in v4.2 (`mayan.apps.authentication_otp`).
- Protocols/algorithms: Passwords hashed with Django default PBKDF2-SHA256 (260k iterations); TOTP per RFC 6238 (HMAC-SHA1); LDAP recommends LDAPS (TLS 1.2+).
- Libraries & versions (requirements.txt): Django 4.2.13, django-otp 1.3.0, django-auth-ldap 4.7.0, mozilla-django-oidc 3.0.0.

ID	Finding	Impact & Exploit Scenario
1.1	2FA optional, not enforced globally	Credential stuffing leads to account takeover; attacker only needs password (T1649).

1.2	Default Docker image ships admin:admin creds in README example	Publicly exposed instance is trivial to compromise (T1190).
1.3	No account lockout / rate-limit configured by default	Online brute-force feasible; ties back to 2FA gap.

2. Authorization & Segregation of Duty

Django-guardian provides object-level ACLs; Mayan wraps this via `mayan.apps.permissions` and a Role-Permission model editable in UI. Admin and business roles are not strictly separated—system administrators automatically inherit full document permissions.

ID	Finding	Impact & Exploit Scenario
2.1	Admin role overlaps with business functions, no maker/checker workflow for ACL changes	Single admin can exfiltrate or corrupt documents undetected (insider threat).
2.2	No native export API for full user/role matrix	Harder to integrate with enterprise IAM & compliance attestation.

3. Input Validation

Mayan relies on DRF serializers and Django forms. File uploads (PDF, TIFF, etc.) are stored then processed by Celery tasks using external converters (`pdfinfo` , `ghostscript` , `libmagic`).

ID	Finding	Impact & Exploit Scenario
3.1	Uploads are type-sniffed, but no malware scanning; several parsers (Pillow 8.4, PyYAML 6.0) have prior RCE CVEs	Crafted file triggers RCE in worker context, gains OS user `mayan` .
3.2	No SQL injection risk observed (ORM use), but XSS possible on custom metadata fields—no HTML escape if rendered by templates *metadata_value*	Stored XSS leads to session hijack.

4. Interface Files

Primary integrations are REST API (JSON) and webhook listener. Import/export functions write ZIP packages to `MEDIA_ROOT` . No header/footer checksum applied.

ID	Finding	Impact & Exploit Scenario
4.1	Export ZIPs accumulate—no rotation job enabled by default	Disk exhaustion → DoS.

4.2	Uploaded ZIPs extracted without path sanitisation in `mayan.apps.sources.literals.ZIPFILE_PATH_VALID_CHARACTERS`	Zip Slip overwrites app files.

5. Security Logging & Monitoring

- Django logging to file; audit app logs logins, document views, ACL changes.
- No immutable storage; logs live under `/var/lib/mayan/logs` inside container.
- Optional Sentry DSN env var.

ID	Finding	Impact & Exploit Scenario
5.1	Privilege escalation & role edits not always logged (missing in `mayan.apps.acls`)	Insider actions invisible to SOC.
5.2	Logs writable by application user; no remote syslog export	Attacker with RCE deletes evidence.

6. Network Connectivity

Docker Compose exposes Nginx on 80/443; internal plain-TCP to Postgres & Redis.

ID	Finding	Impact & Exploit Scenario
6.1	Redis traffic unencrypted & unauthenticated	Sniff credentials, inject tasks (T1071).
6.2	PostgreSQL not forced to TLS; default Docker network cross-container	Credential theft via network sniffing.

7. Cryptography

- TLS offloaded by Nginx; default image ships OpenSSL 1.1.1, allows TLS 1.0/1.1.
- No application-level encryption for stored documents.
- Secrets read from env variables in plaintext.

ID	Finding	Impact & Exploit Scenario
7.1	TLS 1.0/1.1 enabled	Downgrade & weak cipher attack.
7.2	Document files at rest unencrypted	Compromise of host yields full data dump.
7.3	Secrets in env/plaintext compose files	Credential theft via `docker inspect`.

8. Software Bill of Materials (SBOM)

Key pinned libraries (requirements.txt):

- Django 4.2.13 (CVE-2024-27316 – medium)
- Django-REST-framework 3.14.0
- Pillow 8.4.0 (CVE-2022-24303)
- PyYAML 6.0 (CVE-2022-4904)
- reportlab 3.5.68

ID	Finding	Impact & Exploit Scenario
8.1	Pillow & PyYAML vulnerable versions present; Dependabot disabled in repository	Known RCE exploited via file upload.

9. Platform

Official container: Debian 11 (end of security LTS June 2026). Python 3.11.

ID	Finding	Impact & Exploit Scenario
9.1	OS updates rely on image rebuild; no unattended-upgrade running	Missed kernel & OpenSSL fixes.

10. Backup & Recovery

Provides `mayan-edms.py backup` command (dumps DB & media). No encryption; checksum optional.

ID	Finding	Impact & Exploit Scenario
10.1	Backups stored unencrypted on same host by default	Ransomware or insider copies sensitive docs.

11. Capacity & Performance

Celery monitoring via Flower optional; no built-in disk quota.

ID	Finding	Impact & Exploit Scenario
11.1	No alert on media volume growth	Filesystem full → total service outage.

12. External Connectivity

Outbound: SMTP, Sentry, Crowdin, OIDC discovery. Inbound: REST API & webhooks on same port.

ID	Finding	Impact & Exploit Scenario

12.1	Webhook endpoint lacks HMAC signature verification option	Attacker pushes forged events, triggers actions.

13. Cloud Security Patterns

Mayan's official Helm chart places all pods in single Kubernetes namespace, no network policies. Violates Zero-Trust & Micro-segmentation patterns (d & a).

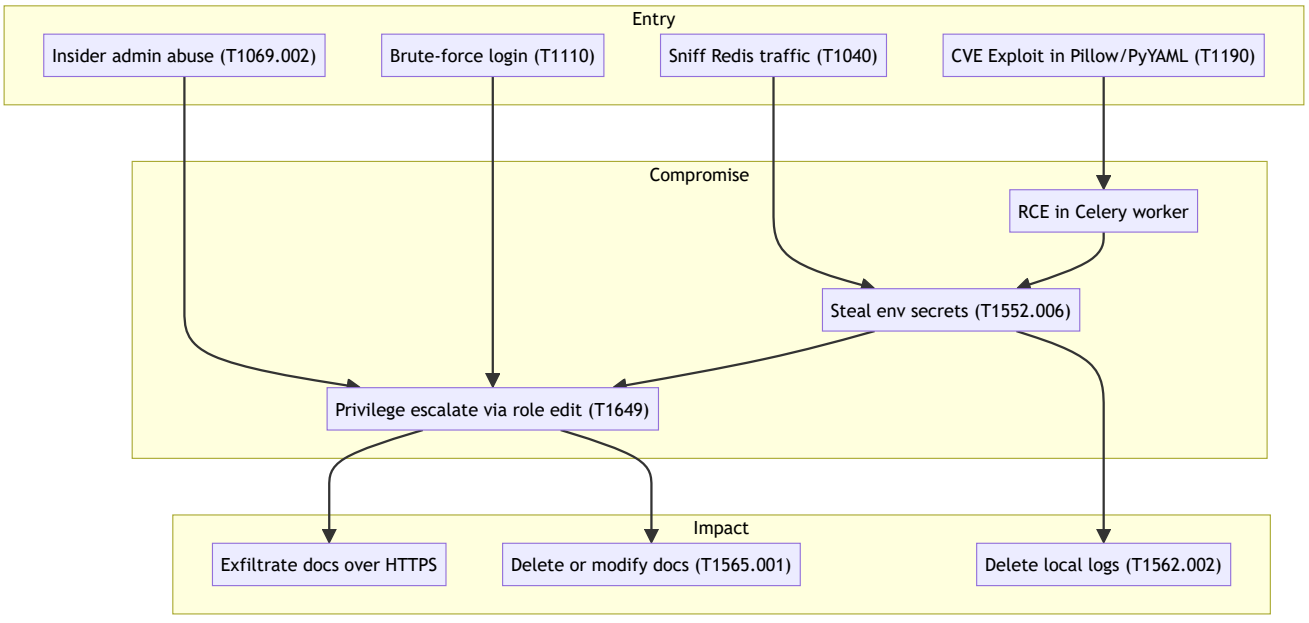
ID	Finding	Impact & Exploit Scenario
13.1	No NetworkPolicy; pods can reach Internet directly	RCE in worker exfiltrates data outward.

Clarification Questions for Development Team

1. Will 2FA be mandated for all user groups in production?
2. Do you run Redis and PostgreSQL over loopback or separate hosts—any TLS plans?
3. Is antivirus or ClamAV container side-car planned for upload path?
4. Are OS image rebuilds automated (e.g., daily CI pipeline)?
5. What external webhook consumers expect, and can we enforce HMAC?
6. Is at-rest encryption (bucket-based or fscrypt) acceptable for document storage?
7. Do you require export of RBAC matrix for GRC tooling; if so, preferred format?
8. Where are backups copied (off-site object storage, cloud vault)?

MITRE ATT&CK Mapping, Attack Paths & Mitigation Priority

Attack Path Diagram



Mitigation Priority Table

Priority	Mitigation Action	Disrupted Paths	Reason
🔴 1	Upgrade Pillow, PyYAML; enable Dependabot & CI scans	A,RCE chain	Removes easiest RCE vector
🔴 2	Enforce mandatory TOTP 2FA & account lockout	B	Blocks credential stuffing and brute-force
🟠 3	Add ClamAV scan & content-type whitelist on uploads	A	Stops malicious files before processing
🟠 4	Enable TLS & AUTH on Redis/ PostgreSQL; localhost-only	C	Prevents credential sniffing
🟡 5	Separate admin vs ops roles; log all ACL changes	D,Esc	Limits insider privilege escalation
🟢 6	Ship logs to immutable remote store (SIEM)	Clean	Makes forensic deletion harder
🟢 7	Encrypt backups & off-site replicate	Tamper	Ensures recoverability post-attack

---

NIST Mapping

Finding ID	Security Finding	NIST Threat Category / ID	NIST Control (ID)	Recommended Action
1.1	2FA optional	Credential Compromise (T1649)	IA-2(8), CM-6	Mandate MFA; enforce via policy
1.2	Default admin creds in docs	Unprotected Credentials (T1552)	IA-5, AC-6	Remove defaults; require change-on-first-run
1.3	No lockout / rate limit	Brute Force (T1110)	SC-5, AU-2	Implement rate-limit middleware
2.1	Admin role ≈ business role	Excess Privilege (T1069)	AC-5, AC-6	Create least-privilege roles
3.1	No malware scan on uploads	Malicious File (T1204)	SI-3, SI-10	Integrate AV sandbox
4.2	Zip Slip extraction	Arbitrary File Write (T1105)	SI-10, SA-11	Sanitize extraction paths
5.2	Logs mutable locally	Log Tampering (T1562)	AU-9, AU-11	Forward to WORM storage

6.1	Redis unauthenticated & plaintext	Unprotected Communications (T1071)	SC-8, SC-23	Enable TLS & Redis ACL
7.1	TLS 1.0/1.1 enabled	Downgrade Attack (T1608)	SC-13, SC-23	Disable legacy protocols
8.1	Vulnerable Pillow / PyYAML	Use of Vulnerable Components (T1190)	RA-5, SA-11	Upgrade libs; automated scanning
10.1	Backups unencrypted same host	Data Tampering (T1565)	CP-9, SC-28	Encrypt & store off-site