ChatGPT

Executive Summary: Das Sicherheitsniveau der Anwendung ist **kritisch**. Zwar wurden positive Ansätze wie Datei-Verschlüsselung und MFA-Unterstützung umgesetzt, jedoch existieren mehrere schwerwiegende Schwachstellen. Insbesondere mangelhafte Zugriffskontrollen und Standard-Passwörter ermöglichen einem Angreifer potenziell die Übernahme des Systems oder unautorisierten Datenzugriff. Auch die Code-Qualität leidet unter Anti-Pattern (z.B. sehr große View-Funktionen), was die Wartung erschwert und Sicherheitslücken begünstigt. Nachfolgend eine detaillierte Aufstellung der Befunde:

Schweregrad	Art der Schwachstelle	Ort (Datei & Zeile)	Beschreibung	Remediation (Lös

Kritisch	Broken Access Control (IDOR)	*dms/views.py* – Bulk Edit (`bulk_edit_documents` bei ca. Zeile 820) [1] [2]	Die Bulk-Bearbeitung von Dokumenten prüft keine berechtigungsspezifischen Zugriffe auf die übergebenen Dokument-IDs. Jeder eingeloggte Benutzer kann Dokumente bearbeiten oder neu zuweisen, selbst wenn er dafür keine Rechte hat. Im Code werden alle übermittelten IDs ungefiltert geladen [1]; nur beim Lösch-Vorgang erfolgt eine Rechteprüfung, andere Aktionen (Status ändern, Mitarbeiter zuordnen, Tags) sind ungeschützt [2]. Dies ermöglicht unautorisierte Änderungen an fremden Dokumenten (z.B. Zuweisung eines Dokuments zu sich selbst).	**Lösung:** Zugriffsl
				Zum einen sollte
				aufgerufen werd
				`@permission_r`
				Zum anderen mu
				Dokumente auf s
				aktuelle Nutzer Z
				vor dem Ändern
				`documents.filt`
				`Q(employee__in`
				die der Nutzer ke
				ein 403-Fehler zu
				stellen sicher, das
				werden können.

Schweregrad	Art der Schwachstelle	Ort (Datei & Zeile)	Beschreibung	Remediation (Lös
Kritisch	Unsichere Voreinstellung (Standard-Admin)	*entrypoint.sh* – Admin-User-Erstellung (Zeile 25–33) [3]	Beim Start wird automatisch ein Admin-Benutzer mit fest kodierten Standard-Anmeldedaten angelegt (`admin` / `admin123`), falls keine Umgebungsvariablen gesetzt sind [3]. Diese schwachen Default-Credentials stellen ein enormes Sicherheitsrisiko dar – ein Angreifer könnte sich damit sofort als Administrator anmelden.	**Lösung:** Keine fe Code sollte statto generieren oder Zugangsdaten ve Passwort **nicht** a Skript (`initial` Admin-Passwort Wichtig ist, dass `"admin123"` ak
Hoch	Broken Access Control (Info Disclosure)	*dms/views.py* – Volltextsuche (`fulltext_search`, ca. Zeile 929–943) [4] [5]	Die Volltextsuche liefert Ergebnisse über alle Dokumente, ungeachtet der Benutzerberechtigung. Der Query sucht in *allen* Dokumenten (kein Filter auf den Eigentümer/ zugriffsbeschränkte Dokumente) [4]. Dadurch kann jeder authentifizierte Nutzer Dateititel, Namen von Mitarbeitern oder Dokumenttypen aus der gesamten Datenbank einsehen [5]. Auch die Auto-Vervollständigung (`suggestions`) berücksichtigt keine Rechte. Dies stellt eine unzulässige Datenoffenlegung dar.	**Lösung:** Die Such beschränkt werd kann durch Wied Dokumentenliste Erstellen des Suc z.B.: `if not request.user.` `queryset = qu` `|` `Q(employee__t` Nur so stellt man den Nutzer fremd Auto-Vervorschlä deaktiviert werde

Schweregrad	Art der Schwachstelle	Ort (Datei & Zeile)	Beschreibung	Remediation (Lös
Hoch	Broken Access Control (IDOR)	*dms/views.py* – `personnel_file_detail` (Zeile 642–650) [6]	Im Personalakten-Detail werden *alle* „unzugeordneten“ Dokumente (Status='UNASSIGNED') angezeigt [6], selbst wenn sie von anderen Benutzern hochgeladen wurden. Ein Benutzer, der zumindest Zugriff auf irgendeine Personalakte hat, sieht dadurch bis zu 50 unzugeordnete Dokumente aus dem Posteingang – was potentielle vertrauliche Inhalte fremder Dokumente preisgibt. Zudem könnte ein böswilliger Nutzer solche Dokumente seiner eigenen Personalakte hinzufügen (über die UI), um sich Zugriff auf deren Inhalte zu verschaffen.	**Lösung:** Unzuge berechtigte Rolle Sektion nur Admi anzuzeigen. Im C `Document.obje` einer Berechtigu der aktuelle Nutz `unassigned_do` Dokumente enth Dadurch sieht ei anderer. Alternat Dateien nur über Frontend) erfolge
Hoch	Veraltete Bibliothek (Broken Authentication)	*requirements.txt* – Verwendung von django-mfa3 < 0.5.0 (CVE-2022-24857) [7]	Die Anwendung nutzt MFA (`django-mfa3`), jedoch gibt es in Versionen unter 0.5.0 eine bekannte Schwachstelle: Die 2-Faktor-Authentifizierung wird für den Admin-Login nicht erzwungen. Django’s Admin-Login-Seite wird von django-mfa3 <0.5.0 nicht geschützt, sodass ein Angreifer die MFA umgehen könnte [7]. Ist die Bibliothek veraltet, kann ein Angreifer mit gültigen Zugangsdaten (z.B. dem Standard-Adminpasswort) direkt ins Admin-Interface gelangen, ohne MFA.	**Lösung:** Die MFA aktualisieren (mi Versionen ist der einbezogen. Falls Admin-Login selb Login auf die nor Maßnahmen laut Wichtig ist, dass MFA-Sicherung b

Schweregrad	Art der Schwachstelle	Ort (Datei & Zeile)	Beschreibung	Remediation (Lös
Mittel	Informationsleck durch Logs	*dms/views.py* – `system_logs` (Zeile 979–987) [8]	Der Endpunkt zur Anzeige der System-Logs ist nur mit Login geschützt, aber nicht auf Admins eingeschränkt. Jeder authentifizierte Benutzer kann also die Log-Ausgaben der Anwendung abrufen [8]. Diese Logs können sensible Informationen enthalten, z.B. Dateinamen, Benutzeraktionen oder Fehlermeldungen, die einem gewöhnlichen Nutzer nicht zugänglich sein sollten. Damit besteht ein **Informationsleck** und potenziell eine Grundlage für weitere Angriffe (durch Auswertung von Systemdetails).	**Lösung:** Zugriffs einfachste Maßn entsprechenden `@permission_r` `raise exceptio` freizugeben. Alte `system_logs` oder ein bestimm zurückgeben. So Administratoren
Mittel	Hardcoded Secret (Krypto)	*dms_project/settings.py* – SECRET_KEY Default (Zeile 4–7) [9]	In den Django-Einstellungen ist ein fest codierter Secret Key als Fallback hinterlegt (`'dev-secret-key-change-in-production'`) [9]. Falls die Umgebungsvariable `DJANGO_SECRET_KEY` in Produktion vergessen wird, läuft die Anwendung mit diesem bekannten Schlüssel. Dadurch wären alle kryptographischen Funktionen (Session-Cookies, Token-Signaturen etc.) kompromittiert, da der Key öffentlich bekannt ist.	**Lösung:** In Produ gesetzt sein. Der Live-System verw den Hardcoded-F aufzudecken. Das zufälligen Key für genutzt wird. Prü ein sicherer SECR dokumentieren S

Schweregrad	Art der Schwachstelle	Ort (Datei & Zeile)	Beschreibung	Remediation (Lös
Mittel	Unsichere Voreinstellung (Samba)	*setup.py* – Initiales Samba-Passwort (Zeile 101–107) [10]	Bei der Einrichtung wird eine Samba-Konfigurationsdatei mit dem Benutzer `dmsuser` und dem Passwort **„changeme“** erzeugt [10]. Dieses Passwort ist standardmäßig schwach und bekannt. Sollte es nicht umgehend durch den Administrator geändert werden, könnten Angreifer mit Zugriff auf das Netzwerk diese Anmeldedaten ausnutzen, um auf freigegebene Verzeichnisse zuzugreifen (oder aus der .env-Datei auslesen und damit auf den SMB-Share zugreifen).	**Lösung:** Adminis Zugangsdaten zu Passwort entwed Admin-Account) Code könnte mar Wert anstelle `"c` Administrator an Interface deutlich Standardpasswo
Niedrig	Fehlerbehandlung (Code Smell)	z.B. *dms/views.py* – `upload_file` (Zeile 253–257) [11], `document_page_thumbnail` (Zeile 473–476) [12]	An vielen Stellen werden Exceptions sehr allgemein abgefangen und teils ungeschickt behandelt. Im Upload-Beispiel wird *jede* Exception gefangen und nur eine generische Fehlermeldung zurückgegeben [11] – das verschluckt Fehler und erschwert die Fehlersuche. Umgekehrt gibt die Thumbnail-Generierung den konkreten Fehlertext direkt im HTTP-Response zurück [12] (inkl. `str(e)`), was interne Informationen preisgeben kann. Beide Ansätze sind suboptimal.	**Lösung:** Bessere einführen. Spezif ungültiges Forma Nutzer als verstä Unbekannte Exce loggen (für Entwi Fehlermeldung g Fehlermeldunger Disclosure verme Behandlung bleib

Schweregrad	Art der Schwachstelle	Ort (Datei & Zeile)	Beschreibung	Remediation (Lös
Niedrig	Performance/DoS Risiko (Best Practice)	z.B. *dms/views.py* – Bulk-Edit Tags Loop (Zeile 865–874) [13], `fulltext_search` count (Zeile 964–969) [14]	Es wurden einige potentielle Performance-Probleme identifiziert, die bei hoher Last zu DoS-Problemen führen könnten. Beispiele: In der Bulk-Bearbeitung wird für jedes Dokument in Python geloopt und Tags einzeln hinzugefügt [13] – das ist ineffizient und belastet die DB unnötig. Die Volltextsuche ruft `results.count()` zweimal auf [14], was bei großen Datenmengen sehr teuer ist. Zudem liest `document_split` komplette PDFs in den RAM, was bei max. 50 MB zwar ok ist, aber bei mehreren gleichzeitigen Prozessen zu Memory-Problemen führen kann. Kein Rate-Limiting für solche Aktionen bedeutet, ein Nutzer könnte die Serverressourcen ausreizen.	**Lösung:** Diese Pu sollten aber optir Operationen der *bulk_update* oder statt in Python-Sc Möglichkeit gestr Verschlüsseln sta Volltextsuche kör oder ein Limit ein entlasten. Zusätz suchintensive En verhindern, dass die Performance

Schweregrad	Art der Schwachstelle	Ort (Datei & Zeile)	Beschreibung	Remediation (Lös
Info	Wartbarkeit (Code Quality)	Gesamtprojekt (Architektur)	Die Codebasis zeigt einige **Wartbarkeitsprobleme**. Insbesondere ist die Logik in wenigen großen Dateien konzentriert – z.B. umfasst *views.py* über 1400 Zeilen mit vielen verschiedenen Verantwortlichkeiten. Doppelter Code (z.B. sehr ähnliche Filterlogik für Dokument- und Aktenlisten) und fehlende Modularisierung erschweren das Verständnis. Solche Monolithen sind fehleranfällig: Änderungen an einer Stelle können unerwartete Nebenwirkungen haben, und Sicherheitslücken werden leicht übersehen.	**Lösung:** Refaktor große Module in umrissenen Vera Apps/Module für Personalakten, Sy Berechtigungspr Middleware ausg gewährleisten. D einer REST-Archit Struktur verbesse zu übersichtliche schneller erkannt

Legende: *IDOR* = *Insecure Direct Object Reference* (unsichere Direktzugriffsreferenz); *Broken Access Control* = fehlerhafte Zugriffskontrolle; MFA = Multi-Faktor-Authentifizierung; DoS = Denial of Service (Dienstverweigerung).

---

1 2 4 5 6 8 11 12 13 14 views.py
https://github.com/hi-its-lukas/sage-local-dms/blob/227ee0077f2122043a19b2a0038673792934736b/dms/views.py

3 entrypoint.sh
https://github.com/hi-its-lukas/sage-local-dms/blob/cffe22357666c74acfa8a0b2977f0005251c12f1/entrypoint.sh

7 CVE-2022-24857.md
https://github.com/trickest/cve/blob/4749005c9ce65092b215e34cac81aa502b04bc28/2022/CVE-2022-24857.md

9 settings.py
https://github.com/hi-its-lukas/sage-local-dms/blob/64e68cfaadad3accc06617423e48ddd17f1ba3a2/dms_project/settings.py

10 setup.py
https://github.com/hi-its-lukas/sage-local-dms/blob/cffe22357666c74acfa8a0b2977f0005251c12f1/setup.py