Penetrationstest der Muster Webapplikation

Ergebnisbericht

KLASSIFIZIERUNG: RESTRICTED

Organisation:	Muster GmbH
An:	Alice Argyle (alice@example.com) Bob Bright (bob@example.com)
Datum:	26. Mai 2026
Version:	1.0
Projekt ID:	2025-06-20
Autor:	

Inhaltsverzeichnis

- Executive Summary
  - Schwachstellenübersicht
  - Schwachstellenverteilung nach Schweregrad
- Testabdeckung
- Methodologie
- Schwachstellen
  - 1. Broken Access Control
  - 2. Cross Site Scripting (XSS)
  - 3. Cross-Site Request Forgery (CSRF)
  - 4. Kein Schutz vor Brute-Force-Angriffen
  - 5. User-Enumeration durch Timing-Angriff
  - 6. Verbesserungswürdige Passwortrichtlinie
  - 7. Open Redirect
  - 8. Content Security Policy fehlt
  - 9. Referrer-Policy Header fehlt
  - 10. X-Frame-Options Header fehlt
- Impressum

Executive Summary

Dieser Bericht beschreibt die Ergebnisse eines Penetrationstests der Muster Webapplikation. Der Umfang des Penetrationstests betrug 5 Personentage (exkl. Berichterstellung). Der Test wurde im *Time-Box*-Verfahren durchgeführt, wobei das Ziel war, innerhalb der zur Verfügung stehenden Zeit möglichst viele Schwachstellen zu identifizieren. Dabei wurden typischerweise risikoreichere Schwachstellen zuerst getestet.

Der Penetrationstest hat eine schwerwiegende Schwachstelle aufgedeckt die zum Verlust der Vertraulichkeit führt: Jeder angemeldete User kann die privaten Profildaten aller anderen Konten der Plattform einsehen. Darunter E-Mail-Adressen und persönliche Einstellungen. Dafür sind weder besondere Kenntnisse noch spezielle Werkzeuge erforderlich; es genügt, in der Applikation eingeloggt zu sein. Alle User sind von dieser Schwachstelle betroffen, weshalb sie mit höchster Priorität behoben werden sollte.

Darüber hinaus ermöglicht eine Stored-Cross-Site-Scripting-Schwachstelle, dass eine bösartige Person mit einem Konto beliebigen Code im Browser anderer User und Admins ausführen kann. Damit besteht ein realistischer Weg zur Kontoübernahme, der in Kombination mit anderen Schwachstellen zu weiterreichendem Schaden führen könnte. Auch die Passwortrichtlinie sollte überarbeitet werden: Die aktuellen Anforderungen begünstigen vorhersehbare Passwortmuster statt wirklich starker Passwörter und machen Konten dadurch anfälliger für Angriffe mit bekannten Passwörtern.

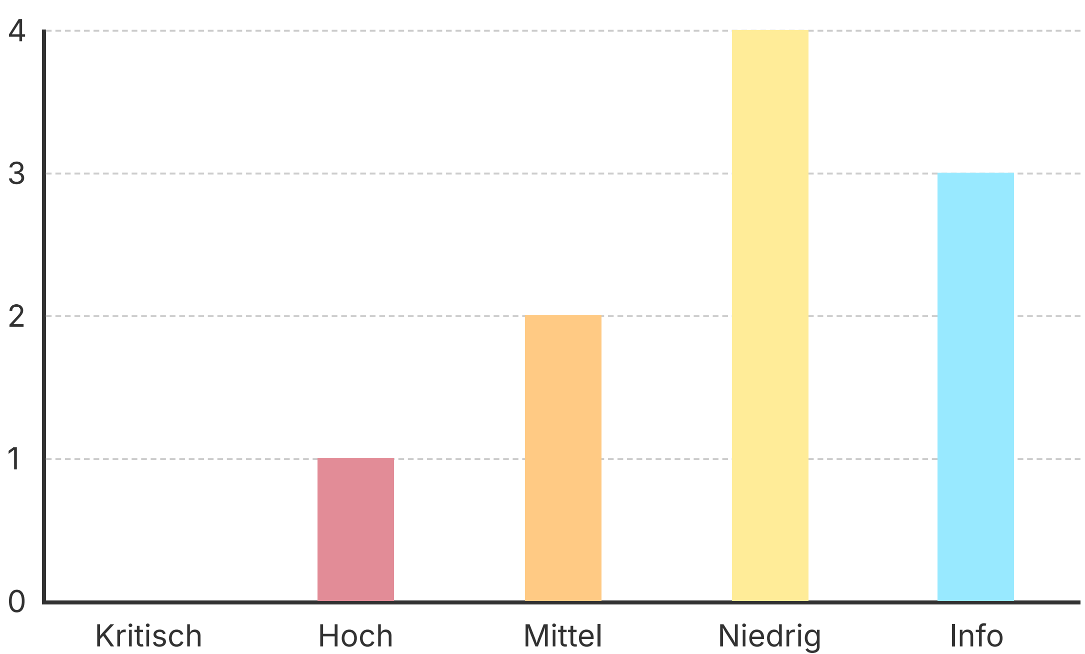
Die übrigen Schwachstelle sind weniger schwerwiegend und erfordern in den meisten Fällen eine Verkettung mehrerer Bedingungen, um tatsächlichen Schaden anzurichten.

Die Schwachstellen sollten behoben werden. In weiterer Folge empfiehlt es sich die Behebung zu verifizieren und die Applikation regelmäßigen Tests zu unterziehen um das Sicherheitsniveau nachhaltig zu verbessern.

Schwachstellenübersicht

Schwachstelle	Schweregrad
1. Broken Access Control	Hoch
2. Cross Site Scripting (XSS)	Mittel
3. Cross-Site Request Forgery (CSRF)	Mittel
4. Kein Schutz vor Brute-Force-Angriffen	Niedrig
5. User-Enumeration durch Timing-Angriff	Niedrig
6. Verbesserungswürdige Passwortrichtlinie	Niedrig
7. Open Redirect	Niedrig
8. Content Security Policy fehlt	Info
9. Referrer-Policy Header fehlt	Info
10. X-Frame-Options Header fehlt	Info

Schwachstellenverteilung nach Schweregrad



Testabdeckung

Es wurde ein bei der Muster GmbH im Umfang von **5 Personentagen** durchgeführt.

Zur Durchführung des Tests wurden dem Penetrationstester folgende Ressourcen zur Verfügung gestellt:

- 2 Konten mit User-Berechtigungen in der Applikation
- 2 Konten mit Admin-Berechtigungen in der Applikation

Folgende Systeme waren Ziel des Penetrationstests:

System	Beschreibung
example.com	Muster Webapplikation

Der Test wurde im Zeitraum 2026-02-01 – 2026-02-05 durchgeführt.

Methodologie

Dieser Penetrationstest wurde mit einem *Time-Box*-Verfahren durchgeführt. Das bedeutet, dass in der zur Verfügung stehenden Zeit so viele Schwachstellen wie möglich aufgedeckt wurden. Dabei wurde priorisiert vorgegangen und zuerst auf Schwachstellen getestet, die typischerweise problematischer sind.

Für jede Schwachstelle wurde ein technischer Schweregrad ermittelt. Dieser spiegelt nicht immer das tatsächliche Unternehmensrisiko wider, das eine Schwachstelle darstellt. Für die tatsächliche Risikobewertung sollten zusätzlich zum technischen Schweregrad weitere Parameter wie die Wichtigkeit des betroffenen Services berücksichtigt werden. Auf Basis des Risikos können anschließend geeignete Risikobehandlungsmaßnahmen abgeleitet werden.

Schweregrad	Bedeutung
Kritisch	Die Schwachstelle ist sehr problematisch und sollte umgehend behoben werden. Beispielsweise ist darüber die vollständige Übernahme eines kritischen Systems möglich.
Hoch	Die Schwachstelle ist problematisch und sollte schnellstmöglich behoben werden. Beispielsweise ist darüber die Übernahme eines Systems möglich.
Mittel	Diese Schwachstelle kann unter Umständen problematisch sein und sollte analysiert werden. Beispielsweise können darüber sensible Informationen ausgelesen werden.
Niedrig	Diese Schwachstelle sollte analysiert werden. Beispielsweise kann sie in Kombination mit anderen Schwachstellen ausgenutzt werden oder interne Informationen preisgeben.
Info	Diese Art von Schwachstelle ist an sich nicht problematisch. Beispiele wären Maßnahmen, die die Sicherheit noch weiter steigern könnten.

Dieser Bericht benutzt Common Vulnerability Scoring System in Version 4.0 zur Bewertung der Schweregrade der Schwachstellen. CVSS ist Eigentum von

FIRST.Org, Inc. (FIRST), einer gemeinnützigen Organisation mit Sitz in den USA.

In diesem Dokument wird CVSS im Detail beschrieben: https://www.first.org/cvss/v4-0/cvss-v40-specification.pdf

Schwachstellen

1. Broken Access Control

Hoch (7,1)

Betroffene Systeme

- example.com

CVSS-Vektor

CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:H/VI:N/VA:N/SC:N/SI:N/SA:N

Beschreibung

„Broken Access Control“ ist eine Schwachstellenkategorie, bei der User auf eine Funktionalität in der Applikation zugreifen können, auf die sie eigentlich keinen Zugriff haben dürfen. In vielen Fällen handelt es sich hier einfach um Flüchtigkeitsfehler in der Implementierung der Berechtigungen. Das zugrunde liegende Problem ist oft eine unzureichende Dokumentation, wer auf welche Funktionalität Zugriff haben sollte.

„Broken Access Control“ ist auf Platz 1 der OWASP Top Ten [1]. Das heißt, es handelt sich um die häufigste in Webapplikationen auftretende Schwachstellenkategorie. Das liegt auch daran, dass es hier wenig technische Unterstützung zur Behebung gibt. Das Berechtigungsmodell muss von einer Person mit Detailwissen zur Applikation und den Use Cases ausgearbeitet und in Folge fehlerfrei implementiert werden.

Bei Insecure Direct Object Reference (IDOR)[2] und Broken Object Level Authorization (BOLA)[3] handelt es sich um spezielle Ausprägungen von Broken Access Control [4]. Hier kann auf Dateien zugegriffen werden, indem eine (zufällige oder aufsteigende) Objekt-ID erraten wird.

Konkrete Schwachstelle

Ein authentifizierter User kann durch Manipulation der User-ID im API-Pfad auf Profildaten anderer User zugreifen. Ein `GET`-Request an `/api/users/{id}/profile` liefert das vollständige Profil des angegebenen Users — einschließlich E-Mail-Adresse, Anzeigename und Kontoeinstellungen — unabhängig davon, ob der authentifizierte User Eigentümer dieses Kontos ist. Durch sequentielles Durchlaufen numerischer IDs können so die Profile aller registrierten User abgerufen werden.

Gegenmaßnahmen

Im ersten Schritt sollten die konkreten gefunden Fehler in der Zugriffskontrolle behoben werden.

Im nächsten Schritt sollte eine Dokumentation entwickelt werden, welche User bzw. Rollen auf welche Funktionen welche Zugriffsberechtigungen haben. Die hier ausgearbeiteten Berechtigungen müssen nun umgesetzt werden. Im letzten Schritt müssen die Berechtigungen noch von einer unabhängigen Person überprüft werden (z. B. Penetrationstest).

Als fortgeschrittene Maßnahme empfiehlt es sich, die Berechtigungen zusätzlich in einem maschinenlesbaren Format zu dokumentieren. Dadurch ist es möglich, automatisierte Testfälle zu entwickeln, um sicherzustellen, dass Fehler in den Berechtigungen möglichst frühzeitig entdeckt werden.

Als zusätzliche Sicherheitsmaßnahme können IDs zufällig gewählt werden (UUID v4[5]). Dadurch muss zwar immer noch eine Zugriffskontrolle implementiert werden, die Ausnutzung der Schwachstelle wird aber schwieriger, da eine bösartige Person zuerst die zufällige ID eines Objekts herausfinden muss.

---

1. OWASP. OWASP Top Ten: https://owasp.org/www-project-top-ten/
2. OWASP. Insecure Direct Object Reference Prevention Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet.html
3. OWASP. API1:2023 Broken Object Level Authorization: https://owasp.org/API-Security/editions/2023/en/0xa1-broken-object-level-authorization/
4. Mustafa Ergin. What is the difference: BOLA vs IDOR: https://shoutrange.com/insights/bola-vs-idor-what-is-the-difference

5. Wikipedia. Universally unique identifier: https://en.wikipedia.org/wiki/Universally_unique_identifier#Version_4_(random)

2. Cross Site Scripting (XSS) Mittel (6,3)

Betroffene Systeme

- example.com

CVSS-Vektor

CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:P/VC:N/VI:N/VA:N/SC:H/SI:H/SA:N

Beschreibung

Allgemeine Beschreibung

Cross-Site-Scripting (XSS) ist eine Kategorie von Schwachstellen, die bösartigen Personen erlaubt, ausführbaren JavaScript-Code in die Webapplikation einzuschleusen. Über den von der bösartigen Person eingeschleusten JavaScript-Code kann die Webapplikation beliebig manipuliert werden, Aktionen im Kontext eines Opfers ausgeführt werden oder Geheimnisse eines Opfers ausgelesen werden. Ein bekanntes Werkzeug, um Schadcode im Browser des Opfers zu übernehmen, ist beef[1]. Im schlimmsten Fall könnte die verwundbare Website daher von einer bösartigen Person missbraucht werden, um User der Website anzugreifen.

Reflected XSS

Bei einer *Reflected XSS*-Schwachstelle kann eine Anfrage an die Website so manipuliert werden, dass in der Antwort auf die Anfrage ausführbarer JavaScript-Code zurückgegeben wird. Ein Beispiel dafür wäre eine Suche, die die ursprüngliche Suchanfrage wieder ausgibt. Im schlimmsten Fall kann eine bösartige Person einen Link generieren, der die Anfrage absetzt. Beim Öffnen des Links wird also der bösartige Code ausgeführt. Das Opfer muss also nur noch dazu verleitet werden, auf den bösartigen Link zu klicken. Das ist in diesem Fall besonders leicht, weil der Link auf die echte, vertrauenswürdige Website führt.

Stored XSS

Auch bei *Stored XSS* wird JavaScript in die Website eingeschleust. Bei *Stored XSS* kann eine bösartige Person JavaScript-Code persistent in die Website einschleusen. Das heißt, dass der Code auch z. B. in einer Datenbank abgespeichert wird. Ein Beispiel dafür ist ein Gästebuch. Eine bösartige Person könnte einen Gästebucheintrag hinterlassen, der JavaScript beinhaltet. Bei Besuchern des Gästebuches würde in Folge der JavaScript-Code ausgeführt.

Konkrete Schwachstelle

Im Eingabefeld „Anzeigename" der Userprofileinstellungen (`POST /api/profile`) wurde eine Stored-XSS-Schwachstelle identifiziert. Wenn ein User seinen Anzeigenamen auf eine Payload wie `<script>alert(document.cookie)</script>` setzt, wird dieser Code ohne Bereinigung gespeichert und anschließend im Browser jedes Users ausgeführt, der eine Seite aufruft, auf der dieser Name angezeigt wird — z. B. in der Userverwaltung des Admin-Panels oder in Kommentar-Threads des betroffenen Users.

Gegenmaßnahmen

Das Eingabefeld „Anzeigename" muss vor der Speicherung bereinigt und vor der Ausgabe im HTML-Kontext kodiert werden. Das React-Frontend sollte für userseitige Werte ausschließlich Text-Node-Rendering (z. B. `textContent`) statt `innerHTML` verwenden.

Grundsätzlich bieten moderne Web-Frameworks gute Schutzmaßnahmen gegen XSS. Diese sollten verwendet werden. Außerdem sollte im Normalfall Ausgabekodierung, statt der Kodierung der Eingabe als Schutzmaßnahme bevorzugt werden. Dadurch bleiben die Eingaben unverändert, der Code wird bei der Ausgabe aber nicht ausgeführt.

Mehr Details zu Gegenmaßnahmen gegen XSS finden sich in [2].

---

1. beefproject. beef: https://github.com/beefproject/beef
2. OWASP. Cross Site Scripting Prevention Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

3. Cross-Site Request Forgery (CSRF)

Mittel (5,1)

Betroffene Systeme

- example.com

CVSS-Vektor

CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:A/VC:N/VI:N/VA:N/SC:N/SI:N/SA:L

Beschreibung

Bei *Cross-Site Request Forgery* (CSRF) handelt es sich um eine Art von Schwachstelle, bei der eine bösartige Website, die vom Opfer besucht wird, im Hintergrund Anfragen im Namen des Opfers abschicken an die verwundbare Website schicken kann.

Ein Beispiel dafür wäre, dass das Opfer Admin in einer Webapplikation ist. Das Opfer ist in dieser Webapplikation eingeloggt und im Browser ist ein aktives Session-Cookie hinterlegt. Das Opfer besucht nun eine bösartige Website. Diese bösartige Website kann nun im Hintergrund über JavaScript HTTP-Anfragen an die anfällige Webapplikation schicken (z. B. Erstellen eines neuen Kontos in der anfälligen Webapplikation). Der Browser des Opfers schickt je nach Konfiguration und Art der HTTP-Anfrage automatisch das Session-Cookie des Opfers mit, die HTTP-Anfragen werden also im Namen des Opfers ausgeführt.

Die getestete Applikation ist bereits relativ gut gegen CSRF abgesichert, da Anfragen, die einen `application/json` Content-Type erfordern, nicht auf CSRF-Angriffe anfällig sind.

Des Weiteren werden in der Applikation die HTTP-Request-Methoden richtig benutzt. GET-Requests lösen beispielsweise keine Änderungen aus. Das ist wichtig, weil diese Methode besonders anfällig für CSRF ist.

Der Endpunkt `/api/auth/logout` akzeptiert GET-Requests und ist nicht durch ein CSRF-Token geschützt. Da GET-Requests von den zusätzlichen Schutzmaßnahmen durch `SameSite=Lax` ausgenommen sind, kann eine bösartige Person das Opfer durch eine unsichtbare Ressourcenanfrage wie <img src="https://example.com/api/auth/logout"> auf einer beliebigen Seite zum Ausloggen zwingen. Wenn ein angemeldeter User die bösartige Seite aufruft, wird die Sitzung unbemerkt beendet.

Gegenmaßnahmen

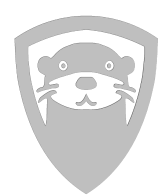
Die klassische Gegenmaßnahme gegen CSRF ist das Setzen von CSRF-Tokens. Dabei generiert die Applikation beim Aufruf der Seite ein zufälliges Token. HTTP-Anfragen müssen dieses Token inkludieren. Für einen erfolgreichen CSRF-Angriff müsste das Token erraten werden.

Zusätzlich muss sichergestellt werden, dass die richtigen HTTP-Methoden verwendet werden. Ein GET-Request darf beispielsweise nie eine Statusänderung in der Applikation auslösen.

Weitere Informationen zum Angriff und empfohlenen Gegenmaßnahmen finden sich in [1]. Die meisten Web-Frameworks bieten Unterstützung bei der Implementierung von CSRF-Gegenmaßnahmen.

---

1. Mozilla. Cross-site request forgery: https://developer.mozilla.org/en-US/docs/Web/Security/Attacks/CSRF



4. Kein Schutz vor Brute-Force-Angriffen

Niedrig (2,3)

Betroffene Systeme

- example.com

CVSS-Vektor

CVSS:4.0/AV:N/AC:H/AT:N/PR:N/UI:P/VC:L/VI:N/VA:N/SC:N/SI:N/SA:N

Beschreibung

Die Anwendung verfügt aktuell über keinen wirksamen Schutz vor Brute-Force-Angriffen.

Dies wurde getestet, indem bei einem einzelnen Konto 100-mal in schneller Folge eine Anmeldung mit falschem Passwort durchgeführt wurde. Alle Anfragen wurden ohne Einschränkung verarbeitet. Unmittelbar nach diesen Versuchen war eine Anmeldung mit korrekten Zugangsdaten erfolgreich. Das bestätigt, dass das Konto weder gesperrt noch durch Rate Limiting geschützt wurde.

Gegenmaßnahmen

Die einfachste Gegenmaßnahme gegen Brute-Force-Angriffe ist eine Kontosperre, bei der ein Konto nach einer festgelegten Anzahl fehlgeschlagener Anmeldeversuche vorübergehend deaktiviert wird. Diese Maßnahme ist wirksam, kann aber für Denial-of-Service-Angriffe (DoS) missbraucht werden, indem legitime Benutzende gezielt ausgesperrt werden.

Alternativ kann Rate Limiting beziehungsweise Request Throttling eingesetzt werden, um die Antwortzeiten nach mehreren fehlgeschlagenen Versuchen künstlich zu erhöhen. Dadurch werden automatisierte Angriffe stark verlangsamt, während legitime Benutzende sich weiterhin anmelden können (mit kurzer Verzögerung).

Ein weitergehender Ansatz nutzt Device-Cookies. Dabei handelt es sich um langlebige Tokens, die bei erfolgreicher Anmeldung erzeugt werden und ein bestimmtes Gerät mit einem Userkonto verknüpfen. Anmeldungen von einem bekannten Gerät mit gültigem Cookie können bestimmte Brute-Force-Schutzmechanismen umgehen, wodurch die Nutzung für legitime Benutzende komfortabler bleibt. Detaillierte Hinweise zur Umsetzung finden sich in [1].

Weitere Informationen zum Blockieren von Brute-Force-Angriffen finden sich in [2].

Letztlich gibt es keinen allgemeingültigen Standard für Brute-Force-Schutz. Die Sicherheitsmaßnahmen sollten immer auf die konkrete Angriffsfläche und das Risikoprofil der Anwendung abgestimmt werden.

---

1. OWASP. Slow Down Online Guessing Attacks with Device Cookies: https://owasp.org/www-community/Slow_Down_Online_Guessing_Attacks_with_Device_Cookies
2. OWASP. Blocking Brute Force Attacks: https://owasp.org/www-community/controls/Blocking_Brute_Force_Attacks

5. User-Enumeration durch Timing-Angriff

Niedrig (2,3)

Betroffene Systeme

- example.com

CVSS-Vektor

CVSS:4.0/AV:N/AC:H/AT:N/PR:N/UI:P/VC:L/VI:N/VA:N/SC:N/SI:N/SA:N

Beschreibung

Obwohl die Anwendung bei vorhandenen und nicht vorhandenen Usern dieselbe generische Antwort zurückgibt, kann durch Vergleich der HTTP-Antwortzeiten festgestellt werden, ob eine E-Mail-Adresse gültig ist.

Password Reset

Diese Schwachstelle wurde beim Passwort-Reset-Formular (`POST /api/auth/reset-password`) festgestellt.

Wenn ein nicht vorhandener User an die Passwort-Reset-Funktion übergeben wurde, lag die Antwortzeit bei ca. 45 ms.

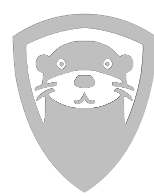
Wenn ein vorhandener User übergeben wurde, lag die Antwortzeit immer über 430 ms, bedingt durch einen synchronen Aufruf des externen E-Mail-Versanddienstes.

Login

Diese Schwachstelle wurde beim Login-Formular (`POST /api/auth/login`) festgestellt.

Wenn ein nicht vorhandener User übergeben wurde, lag die Antwortzeit bei ca. 18 ms.

Wenn ein vorhandener User übergeben wurde, lag die Antwortzeit bei ca. 210 ms, bedingt durch das bcrypt-Passwort-Hashing, das nur für vorhandene



Konten durchgeführt wird.

Beide Ausprägungen dieser Schwachstelle sollten von einem System aus erneut getestet werden, das näher am Zielsystem liegt (z. B. im lokalen Netzwerk), um sicherzustellen, dass kein Hop dazwischen die Antwortzeit beeinflusst hat.

Gegenmaßnahmen

Alle Anfragen sollten ähnliche Antwortzeiten haben, unabhängig davon, ob der angegebene Username existiert oder nicht.

Login

Häufig liegt die Ursache unterschiedlicher Antwortzeiten darin, dass zuerst geprüft wird, ob ein User existiert, und erst danach das Passwort gehasht wird. Als Gegenmaßnahme sollte jedes übergebene Passwort gehasht werden, da das Hashing meist die zeitaufwendigste Operation ist. Erst danach sollte geprüft werden, ob der User existiert. Die Anwendung sollte weiterhin eine generische Fehlermeldung zurückgeben und nicht offenlegen, ob die Prüfung am Passwort oder am Username gescheitert ist.

Password Reset

Oft reagieren Passwort-Reset-Formulare langsamer auf gültige E-Mail-Adressen, weil ein Drittanbieter-Service aufgerufen wird und die Anwendung auf dessen Antwort wartet. Das sollte entweder so geändert werden, dass die Antwort sofort gesendet und der Drittanbieter-Service asynchron aufgerufen wird, oder beide Antworten sollten künstlich verzögert werden, um Timing-Angriffe unmöglich zu machen.

6. Verbesserungswürdige Passwortrichtlinie

Niedrig (2,3)

Betroffene Systeme

- example.com

CVSS-Vektor

CVSS:4.0/AV:N/AC:H/AT:P/PR:N/UI:P/VC:L/VI:L/VA:N/SC:N/SI:N/SA:N

Beschreibung

Eine Passwortrichtlinie soll User dabei unterstützen, ein sicheres Passwort zu setzen, das von bösartigen Personen nicht einfach erraten werden kann.

Aktuell ist folgende Passwortrichtlinie gesetzt:

- Mindestlänge: 8 Zeichen
- Muss mindestens einen Großbuchstaben, einen Kleinbuchstaben, eine Ziffer und ein Sonderzeichen enthalten

Die erzwungenen Komplexitätsanforderungen fördern vorhersehbare Passwortmuster (z. B. `Password123!`), die die Richtlinie erfüllen, Angreifern jedoch bekannt sind. Gleichzeitig existiert keine Blockliste, die die Verwendung solcher häufig genutzten Passwörter verhindert.

Es konnte folgendes Passwort gesetzt werden: `Password123!`.

Gegenmaßnahmen

Das National Institute of Standards and Technology (NIST) hat eine weitläufig akzeptierte Hilfestellung für Passwortrichtlinien herausgegeben (siehe [1]).

Die wichtigsten Punkte daraus sind:

- Mindestens 8 Zeichen (je nach Schutzbedarf kann das auch höher sein)

- Einsatz von modernen Hash-Algorithmen, die speziell für die Speicherung von Passwörtern konzipiert sind (z. B. Argon2)[2]
- Vorhandensein eines Schutzes vor Brute-Force-Angriffen
- Blockliste, die verhindert, dass User bekannte unsichere Passwörter setzen können (z. B. `Password123!`) (dafür kann beispielsweise die Lösung in [3] genutzt werden)
- **Keine** Anforderungen an die Passwortkomplexität
- **Keine** erzwungene regelmäßige Passwortänderung

Wichtig: Damit eine neue Passwortrichtlinie auch wirklich greift, müssen, nachdem die Passwortrichtlinie aktualisiert wurde, alle User dazu gezwungen werden, ihr Passwort zu ändern. Das liegt daran, dass die Passwortrichtlinie nur beim erneuten Setzen des Passworts angewendet wird. Alternativ könnte beim Anmelden auch überprüft werden, ob das aktuell gesetzte Passwort der Passwortrichtlinie entspricht und der User zu einer Änderung aufgefordert werden.

---

1. NIST. Digital Identity Guidelines: https://pages.nist.gov/800-63-3/sp800-63b.html
2. Alex Biryukov, Daniel Dinu, und Dmitry Khovratovich. Argon2: the memory-hard function for password hashing and other applications: https://www.cryptolux.org/images/0/0d/Argon2.pdf
3. Troy Hunt. Pwned Passwords: https://haveibeenpwned.com/Passwords

7. Open Redirect

Niedrig (2,1)

Betroffene Systeme

- example.com

CVSS-Vektor

CVSS:4.0/AV:N/AC:H/AT:N/PR:N/UI:A/VC:N/VI:N/VA:N/SC:L/SI:N/SA:N

Beschreibung

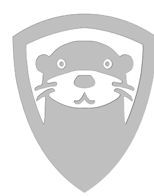
Bei einem Open Redirect handelt es sich um die Schwachstelle, dass die Applikation eine Weiterleitung auslöst, deren Ziel manipuliert werden kann. Dadurch kann eine bösartige Person die Applikation missbrauchen, um ein Opfer auf eine potenziell bösartige Website zu locken. Dabei wird das Vertrauen in die betroffene Webapplikation missbraucht und möglicherweise geschädigt.

Konkrete Schwachstelle

Der Login-Endpunkt (`GET /login`) akzeptiert einen `next`-URL-Parameter, der nach erfolgreicher Authentifizierung als Weiterleitungsziel verwendet wird. Dieser Parameter wird nicht gegen eine Allowlist validiert und akzeptiert beliebige externe URLs. Eine bösartige Person kann einen Link wie `https://example.com/login?next=https://malicious.example` erstellen und als scheinbar legitimen Login-Link verbreiten. Nachdem sich das Opfer auf der echten Login-Seite authentifiziert hat, wird es unbemerkt auf eine von der bösartigen Person kontrollierte Website weitergeleitet, auf der Zugangsdaten oder Session-Token abgegriffen werden könnten.

Gegenmaßnahmen

Es sollte über diese Endpunkte nicht möglich sein eine beliebige externe URL als Ziel der Weiterleitung anzugeben. Falls das ausreichend ist, sollten nur relative, also interne URLs, als Ziel der Weiterleitung erlaubt werden. Sollte das nicht ausreichend sein, sollte eine Allowlist an externen URLs oder Domains erstellt werden, die als Ziel der Weiterleitung erlaubt sind.



Mehr Informationen zur Schwachstelle und möglichen Gegenmaßnahmen finden sich in [1].

---

1. OWASP. Unvalidated Redirects and Forwards Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Unvalidated_Redirects_and_Forwards_Cheat_Sheet.html

8. Content Security Policy fehlt Info (0,0)

Betroffene Systeme

- example.com

CVSS-Vektor

CVSS:4.0/AV:N/AC:H/AT:P/PR:N/UI:P/VC:N/VI:N/VA:N/SC:N/SI:N/SA:N

Beschreibung

Content Security Policy (CSP) ist eine fortgeschrittene Sicherheitsmaßnahme, bei der über einen HTTP-Header definiert wird, von welchen Quellen eine Website externe Ressourcen laden bzw. Code ausführen darf.

Dies erschwert primär die Ausnutzung von Cross-Site-Scripting-Schwachstellen, da standardmäßig nur explizit erlaubter JavaScript-Code ausgeführt werden darf. Inline-Skriptcode oder Code aus unbekannten Quellen wird nicht mehr ausgeführt. Selbst wenn es einer bösartigen Person gelingt, schädlichen JavaScript-Code in die Website einzuschleusen, wird dieser nicht ausgeführt.

Des Weiteren kann darüber beispielsweise auch gesteuert werden, ob die Website in einem `iframe` eingebettet werden darf oder, von welchen Quellen Ressourcen geladen werden dürfen.

Aktuell ist auf dem betroffenen System keine *Content Security Policy* gesetzt.

Gegenmaßnahmen

Die Implementierung einer *Content Security Policy* (CSP) ist nicht trivial, da sie für jede Webapplikation individuell angepasst werden muss. Je nach Aufbau der Webapplikation kann es zudem erforderlich sein, Code zu ändern, um den Einsatz einer CSP zu ermöglichen.

Grundsätzlich sollte eine *Content Security Policy* keine `unsafe`-Attribute inkludieren, da diese wichtige Sicherheitsfeatures deaktivieren.

Details zur Implementierung einer CSP finden sich in [1], [2] und [3].

---

1. Lukas Weichselbaum. Mitigate cross-site scripting (XSS) with a strict Content Security Policy (CSP): https://web.dev/articles/strict-csp
2. Mozilla. Content Security Policy (CSP): https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CSP
3. OWASP. Content Security Policy Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html

9. Referrer-Policy Header fehlt Info (0,0)

Betroffene Systeme

- example.com

CVSS-Vektor

CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:P/VC:N/VI:N/VA:N/SC:N/SI:N/SA:N

Beschreibung

Der *Referrer-Policy*-HTTP-Header bestimmt, ob der Browser beim Öffnen eines Links den `Referer` (also die Website, auf der der Link geklickt wurde) an das Ziel übermittelt. Standardmäßig ist das der Fall.

Das bedeutet, wenn ein User auf der Website `https://quelle.example` einen Link zu `https://ziel.example` klickt, wird der `Referer` `https://quelle.example` im HTTP-Request an `https://ziel.example` mitgeschickt. Dadurch erfährt `https://ziel.example`, dass der User zuvor auf `https://quelle.example` war, was potenziell die Privatsphäre des Users verletzen kann.

Der `Referrer-Policy`-Header war in den HTTP-Antworten von `example.com` nicht gesetzt.

Gegenmaßnahmen

Im Sinne der Datensparsamkeit sollte der *Referrer-Policy*-Header auf `no-referrer`, `origin` oder `same-origin` gesetzt werden. Weitere Informationen zu diesem Header finden sich in [1].

---

1. Mozilla. Referrer-Policy: https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Referrer-Policy

10. X-Frame-Options Header fehlt Info (0,0)

Betroffene Systeme

- example.com

CVSS-Vektor

CVSS:4.0/AV:N/AC:H/AT:N/PR:N/UI:A/VC:N/VI:N/VA:N/SC:N/SI:N/SA:N

Beschreibung

Der *X-Frame-Options*-Header verhindert *Clickjacking*-Angriffe[1]. Dabei bettet eine bösartige Website die anfällige Website in einem *iFrame* ein, das sogar unsichtbar sein kann. Das Opfer wird auf die bösartige Website gelockt und dazu verleitet, auf bestimmte Elemente zu klicken. Da in Wirklichkeit jedoch die anfällige Seite eingebettet ist, führt das Opfer unbemerkt Aktionen auf dieser Seite aus.

Diese Schwachstelle ist besonders problematisch, wenn die anfällige Seite über einen eingeloggten Bereich verfügt, in dem Aktionen durchgeführt werden können. Die getestete Anwendung verfügt über einen solchen eingeloggten Bereich. Allerdings wurden keine Aktionen identifiziert, die durch Clickjacking gezielt ausgelöst werden könnten und dabei eine nennenswerte Auswirkung hätten. Daher wird die Schwachstelle als informationell eingestuft. Es wird dennoch empfohlen, diesen Header als Defense-in-Depth-Maßnahme zu setzen.

Gegenmaßnahmen

Der *X-Frame-Options*-Header sollte auf `DENY` oder `SAMEORIGIN` gesetzt werden[2]. Zusätzlich kann in der *Content Security Policy* die `frame-ancestors`-Direktive gesetzt werden, die eine ähnliche Schutzwirkung hat.[3]

---

1. Mozilla. Clickjacking: https://developer.mozilla.org/en-US/docs/Web/Security/Attacks/Clickjacking
2. Mozilla. X-Frame-Options: https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/X-Frame-Options
3. Mozilla. CSP: frame-ancestors: https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Content-Security-Policy/frame-ancestors

Impressum

VidraSec e.U. – https://www.vidrasec.com/
Anschrift: Almesbergerweg 3, 4160 Aigen-Schlägl, AUSTRIA
Firmenbuchnummer: 623204b
Firmenbuchgericht: Landesgericht Linz
Gewerbeaufsichtbehörde: BH Rohrbach
UID-Nummer: ATU80334229

Kontaktdaten
E-Mail: martin@vidrasec.com
Telefon: +43 670 3081275

Bankverbindung
Bank: REVOLUT BANK UAB
IBAN: LT76 3250 0126 5399 4118
BIC: REVOLT21