HINWEIS

Die Softwarelösung der analysierten Internetpräsenz richtet sich an Organisationen, die Prozesse rund um geistiges Eigentum (Intellectual Property, IP) digital verwalten und strukturieren möchten. Sie unterstützt bei der Erfassung, Bearbeitung und Nachverfolgung von Schutzrechten wie Patenten, Marken oder Designs.

Beobachtungen zur Frontend-Performance

Auffälligkeit #1

Kategorie: Offenlegung von Server-Technologien

Schweregrad: mittel

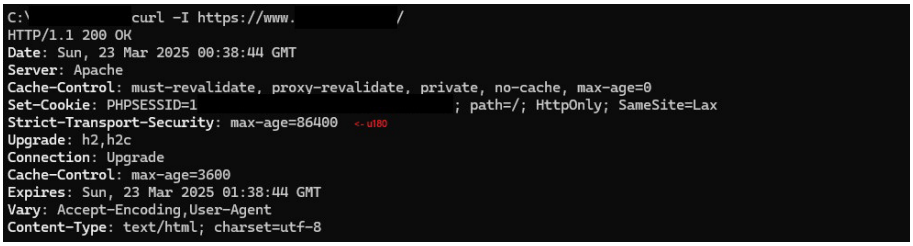
Potentielles Risiko:

Der HTTP-Header **server: Apache** offenbart die zugrunde liegende Webserver-Software. Dies liefert einem potenziellen Angreifer wertvolle Informationen über mögliche Angriffsvektoren, insbesondere wenn der Server oder verwendete Module nicht aktuell sind.

Beschreibung:

Durch die Offenlegung des Servertyps (Apache) kann ein Angreifer gezielt nach bekannten Schwachstellen suchen. In Kombination mit anderen Faktoren (z. B. veraltete PHP-Versionen oder unsichere Konfigurationen) erhöht dies die Angriffsfläche.

Quelle:



Lösungsansatz:

Konfiguration so anpassen, dass der Server-Header entfernt oder anonymisiert wird:

```
# Apache Konfigurations-Datei

ServerTokens Prod
ServerSignature Off
Header unset Server
```

Wichtig: Entsprechende Module hinzufügen o.a. mod_headers

Best Practice: Nginx verwenden. Erheblicher Performance Boost und deutlich bessere konfigurations Möglichkeiten.

Auffälligkeit #2

Kategorie: Exponierte Sessions

Schweregrad: mittel

Potentielles Risiko:

Der Session-Cookie **PHPSESSID** ist weiterhin aktiv und verwendet den Standardnamen. Das erhöht die Vorhersehbarkeit und potenziell die Angriffsfläche bei Anwendungen. Zwar sind HttpOnly und SameSite=Lax gesetzt – jedoch fehlt das Secure-Flag, was die Übertragung über unverschlüsselte HTTP-Verbindungen zulassen würde (sofern TLS nicht forciert wird).

Beschreibung:

Secure-Flag fehlt. Cookie könnte theoretisch über unverschlüsselte Verbindungen übertragen werden

Quelle:

Lösungsansatz:

Secure-Flag setzen, damit der Cookie nur über HTTPS übertragen wird:

```
# Session-Verwaltung
session_set_cookie_params([
   'secure' => true,
   'httponly' => true,
   'samesite' => 'Lax'
]);
```

Session-Namen anpassen, um Angriffe wie Session Fixation oder einfache Session-Erkennung zu erschweren:

```
# php.ini
session.name = CustomSessionID

# php.ini Secure Cookie Opt
session.cookie_httponly = 1
session.cookie_secure = 1
session.cookie_samesite = Lax

# Session-Verwaltung
session_name('CustomSessionID');
```

Auffälligkeit #3

Kategorie: Keine CORS-Richtlinie

Schweregrad: mittel

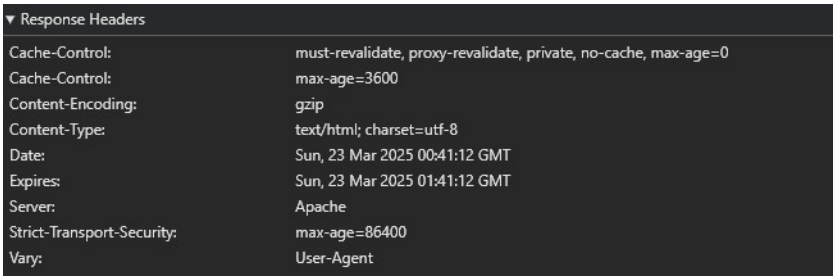
Potentielles Risiko:

Die Seite erlaubt keine Cross-Origin-Requests. Das ist grundsätzlich sicher, kann aber in komplexeren Architekturen zu Problemen führen. Zudem zeigt es, dass CORS nicht aktiv verwaltet wird – ein Versäumnis, das später zu Fehlkonfigurationen führen kann.

Beschreibung:

CORS-Header wie Access-Control-Allow-Origin fehlen vollständig. Zwar ist das sicherer als eine offene Konfiguration, es deutet jedoch darauf hin, dass CORS nicht bewusst kontrolliert wird – ein Risikofaktor bei der aktuellen Infrastruktur oder späteren API-Integrationen.

Quelle:



▼ Response Headers

Cache-Control:	must-revalidate, proxy-revalidate, private, no-cache, max-age=0
Cache-Control:	max-age=3600
Content-Encoding:	gzip
Content-Type:	text/html; charset=utf-8
Date:	Sun, 23 Mar 2025 00:41:12 GMT
Expires:	Sun, 23 Mar 2025 01:41:12 GMT
Server:	Apache
Strict-Transport-Security:	max-age=86400
Vary:	User-Agent

Lösungsansatz:

CORS bewusst setzen oder durchsetzen:

```
# Apache Konfigurations-Datei
Header set Access-Control-Allow-Origin "https://example.com"
Header set Access-Control-Allow-Methods "GET, POST"
Header set Access-Control-Allow-Headers "Content-Type"
```

Auffälligkeit #4

Kategorie: Kein CSP (Content-Security-Policy) Header

Schweregrad: kritisch

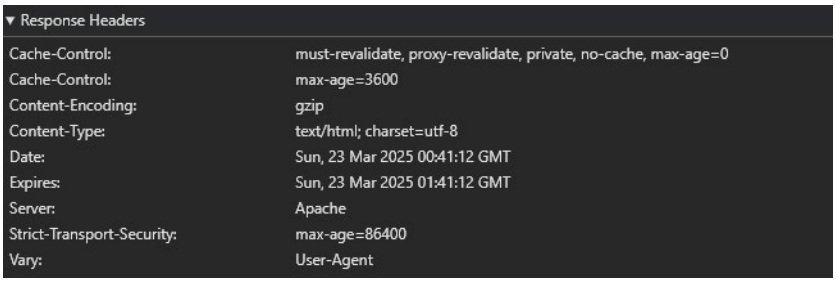
Potentielles Risiko:

Ohne CSP besteht ein erhöhtes Risiko für XSS-Angriffe, da der Browser keine Einschränkungen für eingebettete Skripte, Styles oder Drittinhalte hat.

Beschreibung:

Eine Content-Security-Policy (CSP) ist ein effektives Mittel gegen viele Angriffsarten. Ihre Abwesenheit ist ein Hinweis auf fehlende moderne Sicherheitsmaßnahmen.

Quelle:



Lösungsansatz:

Eine strikte CSP einführen, z. B.:

```
# Apache Konfigurations-Datei
Content-Security-Policy:
  default-src 'self';
  script-src 'self' 'nonce-abc123' 'sha256-xyz='
https://cdn.example.com;
  style-src 'self' 'nonce-xyz';
  img-src 'self' data:;
  connect-src 'self' https://api.example.com;
  font-src 'self' https://fonts.gstatic.com;
  object-src 'none';
  frame-ancestors 'none';
  base-uri 'self';
  form-action 'self';
```

Auffälligkeit #5

Kategorie: Client-Security

Schweregrad: kritisch

Potentielles Risiko:

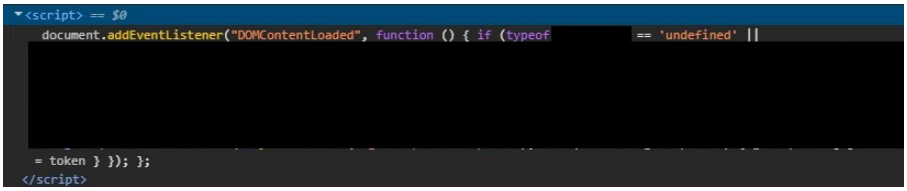
Cross-Site Scripting (XSS), Code Injection
OWASP Top 10 A03:2021 – Injection

Das eingebettete Inline-Skript wird beim Seitenaufruf direkt im HTML ausgeführt und lädt zusätzlich externen JavaScript-Code (Google reCAPTCHA). In der aktuellen Konfiguration existiert keine Content-Security-Policy (CSP) mit Nonce oder Hash, was bedeutet, dass potenziell beliebiger Inline-JavaScript-Code ausgeführt werden kann, wenn er erfolgreich injiziert wird (z. B. durch Reflected-XSS).

Beschreibung:

Die Nutzung eines ungeschützten Inline-Skripts stellt eine potentielle Angriffsfläche für XSS-Angriffe dar. Ohne eine restriktive CSP (z. B. script-src 'self' 'nonce-xyz') oder ein Hash-basiertes Whitelisting (sha256-...) ist der Browser gezwungen, alle Inline-Skripte zuzulassen. In Kombination mit dynamisch nachgeladenem Code aus Drittquellen (z. B. Google) wird der Angriffsvektor zusätzlich erweitert. Dies widerspricht modernen Sicherheitsstandards (z. B. OWASP A03:2021 – Injection).

Beispiel Quelle:



Lösungsansatz:

Das Skript in eine separate .js-Datei auslagern, die von der eigenen Domain geladen wird oder CSP mit nonce- oder sha256-Mechanismus implementieren, um die Ausführung gezielt erlaubter Inline-Skripte abzusichern.

```
# Beispiel
<script nonce="abc123">...</script>
```

Auffälligkeit #6

Kategorie: Hosting Software

Schweregrad: hochkritisch

Potentielles Risiko:

Zugriff auf hochkritische Daten sind potentiell offengelegt.

Beschreibung:

Durch nähere Beobachtung des Traffics bin ich auf ***.com** gestoßen. Nach Recherche konnte ich herausfinden, dass es sich hierbei um ein Admin-Tool des Hosting Providers ***.com** handelt.

Jenachdem welche Daten sich hinter dem *TOOL* befinden, kann das zu etwaigen Problemen führen, da hier keinerlei sicherheitsrelevante Vorkehrungen getroffen wurden. Ganz im Gegenteil.

Hinsichtlich CSP und XSS-Exploits.

Beispiel Quelle:

▼ Response Headers

Cache-Control:	no-cache,must-revalidate,no-store,max-age=0,post-check=0,pre-check=0
Content-Encoding:	br
Content-Security-Policy:	default-src 'self'; img-src 'self' data:; style-src 'self' 'unsafe-inline'; script-src 'self' 'unsafe-inline' 'unsafe-eval' blob: ;
Content-Type:	text/html
Date:	Sun, 23 Mar 2025 01:44:14 GMT
Expires:	-1
Feature-Policy:	none
Pragma:	no-cache
Referrer-Policy:	same-origin
Server:	nginx
Strict-Transport-Security:	max-age=15768000
Vary:	Accept-Encoding,User-Agent
X-Content-Type-Options:	nosniff
X-Frame-Options:	SAMEORIGIN
X-Xss-Protection:	1; mode=block

Lösungsansatz:

Bei Bedarf Provider wechseln

Positive Beobachtungen im Ausgangsprojekt

POST Formulare werden mit Honeypot und CSFR-Token versehen.
SSL Cipher und all. Konfiguration ist sehr sicher.