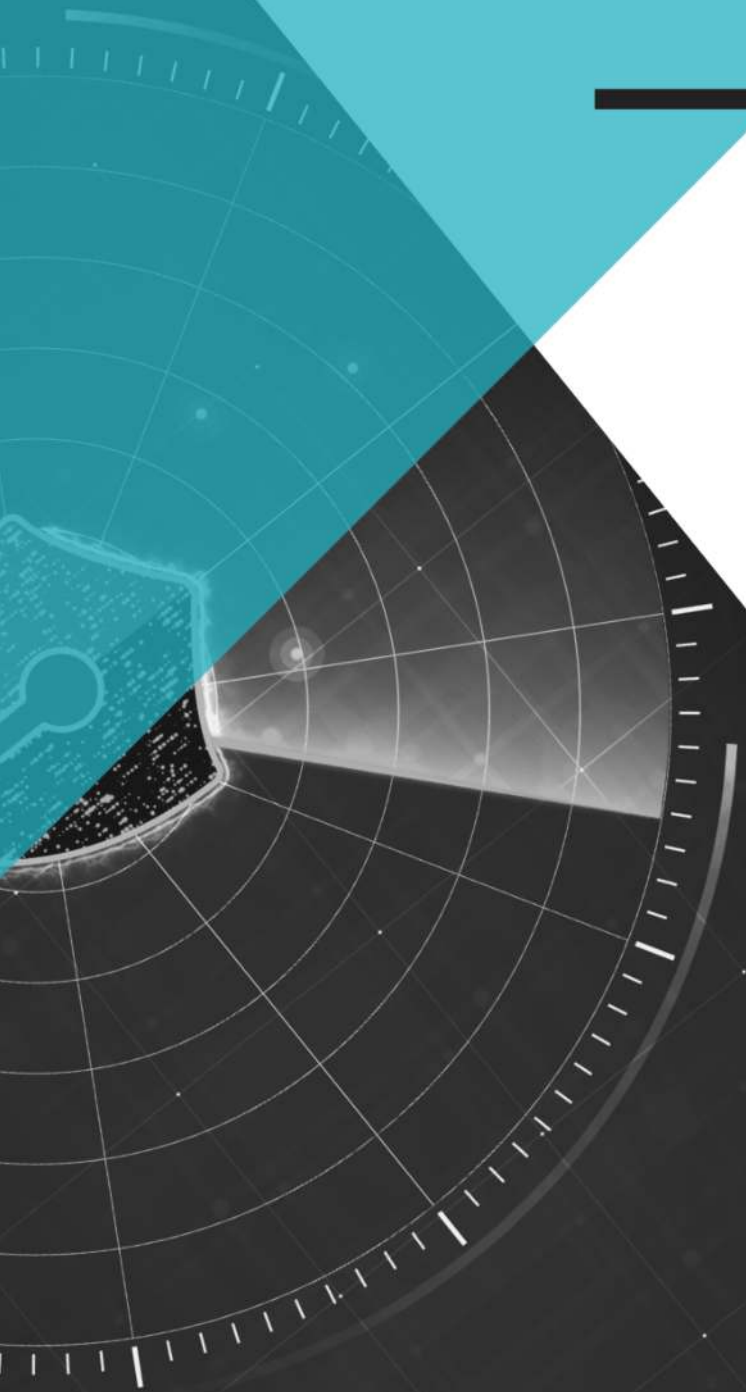
AWARE7

Bericht zur Durchführung eines

Penentrationstests

Franken Logistik GmbH



Dieser Report enthält sensible Informationen über technische und infrastrukturelle Schwachstellen und MUSS daher in der Organisation verbleiben, die diesen Bericht beauftragt hat.

Inhaltsverzeichnis

1 Allgemeine Informationen

Dieses Kapitel enthält allgemeine Informationen rund um den durchgeführten Penetrationstest bei der Franken Logistik GmbH. In diesem Kapitel finden Sie neben Informationen zu dem beauftragten Projekt, eine Dokumentenhistorie, einen Nachhaltigkeitshinweis sowie eine Beschreibung der Risikoklassen, nach denen Schwachstellen in diesem Bericht klassifiziert werden.

1.1. Beauftragtes Projekt

Im Folgenden sind allgemeine projektspezifische Informationen wie Testzeiten und Projektmitarbeiter tabellarisch dargelegt.

Auftraggeber	Franken Logistik GmbH Hafenstraße 4 96047 Bamberg
Projektvolumen	11 PT
Testzeitraum:	21.07.2021 - 06.08.2021
Projektverantwortlicher:	Matteo Große-Kampmann
Projektmitarbeiter:	Zaphod Beeblebrox Lisbeth Salander Thomas A. Anderson Elliot Alderson

Tabelle 1.1: Informationen zum durchgeführten Penetrationstest

1.2. Dokumentenhistorie

Um den zeitlichen Ablauf zur Anfertigung des Berichts zur Durchführung des Penetrationstests nachvollziehen zu können, finden Sie die folgende Tabelle 1.2 vor. Diese beschreibt die Historie zum aktuellen Zeitpunkt.

Datum	Autor	Version
28.07.2021	Zaphod Beeblebrox	V 0.1 Initiale Erstellung und Konfiguration
05.08.2021	Thomas A. Anderson	V 0.2 - Einarbeitung der Findings
06.08.2021	Lisbeth Salander & Elliot Alderson	V 0.3 - Korrekturdurchlauf
06.08.2021	Matteo Große-Kampmann	V 1.0 - Finalisierung und Abnahme

Tabelle 1.2: Dokumentenhistorie

1.3. Projektgegenstand

Der Franken Logistik GmbH beauftragte die AWARE7 mit der Schwachstellennalyse der mobilen-App und Webwanwendung.
Da es sich bei der beauftragten Untersuchung um einen Greybox-Penetratiostest handelte, wurden von Seite der Auftraggeberin die Systeme im Scope klar definiert. Alle Tests wurden auf expliziten Testsystemen durchgeführt, um den produktiven Betrieb der Anwendung zu gewährleisten. Die durchgeführten Tests wurden auf die unten aufgeführten Systeme durchgeführt.
Jedes System, welches nicht in der Tabelle 1.3 aufgelistet ist, ist nicht Teil der Untersuchung.

System	Kommentar
api.example.com	RESTful API
www.example.com	Webseite
example.example.com	Resource Provider
com.example.app	Android App

Tabelle 1.3: Umfang der untersuchten Systeme

1.4. Risikoklassen

Die im Rahmen des Penetrationstests identifizierten Schwachstellen werden anhand ihres Risikograde klassifiziert. In der folgenden Tabelle 1.4 werden die verschiedenen Risikoklassen vorgestellt sowie jeweils angemessene Behandlungsvorgaben und deren Aufwand dargelegt.

Risiko	Behandlungsvorgaben	Aufwand
Kritisch	Diese Risiken gefährden Ihren laufenden Geschäftsbetrieb. Im Rahmen des Tests haben wir diese Risiken bereits an Sie eskaliert.	Es müssen teilweise spontane, sehr hohe Aufwände für die Risikobehandlung in Kauf genommen werden.
Hoch	Das Risiko muss kurzfristig behandelt werden.	Für die Risikobehandlung müssen hohe Aufwände in Kauf genommen werden.
Mittel	Das Risiko sollte mittelfristig behandelt werden.	Maßnahmen zur Risikobehandlung sollten moderaten Mehraufwand verursachen.
Gering	Das Risiko kann in der Regel akzeptiert werden.	Maßnahmen zur Risikobehandlung sollten einen geringen Mehraufwand verursachen.
Information	Wir konnten keine Kategorisierung vornehmen. Sie sollten diese Risiken in Bezug auf Ihre Architektur bewerten.	Maßnahmen zur Risikobehandlung sollten einen geringen Mehraufwand verursachen.

Tabelle 1.4: Übersicht über die Risikoklassen

1.5. Nachhaltigkeitshinweis

Um die Umwelt zu schonen stellen wir Ihnen diesen Bericht ausschließlich digital zur Verfügung. Gerne drucken wir Ihnen den Bericht aus und senden Ihnen ein Exemplar zu, aus Gründen der Nachhaltigkeit verzichten wir jedoch standardmäßig auf einen Ausdruck.

2

Zusammenfassung der identifizierten Schwachstellen

Im folgenden Kapitel werden die Schwachstellen, welche während der Untersuchung der Franken Logistik GmbH identifiziert wurden, zusammengefasst. Dabei werden alle identifizierten Schwachstellen einmal für das Management und einmal technisch zusammengefasst.

2.1. Zusammenfassung für das Management

Die Franken Logistik GmbH beauftragte die AWARE7 GmbH mit einer Sicherheitsanalyse ihrer Web- und Mobileanwendung, bestehend aus nachfolgenden Systemen. Der Umfang des Tests bestand darin, diese drei Systeme und die dazugehörige App, die drei hierarchische Rollen anbieten, zu untersuchen. Der Angriff wurde von extern auf die unten aufgeführten Systeme durchgeführt. Jedes andere System, das auf der Domäne "example.com" gehostet wird, ist nicht Teil der Untersuchung.

Die Untersuchung wurde nach dem aktuellen Stand der Technik und durch fachkundiges Personal durchgeführt. Die Tests wurden mit größtmöglicher Sorgfalt und nach bestem Gewissen durchgeführt. Als Testmethodik diente die aktuelle Version des Schwachstellenkatalogs vom Open Web Application Security Projekt (OWASP). Wenn für eine entsprechende Kategorie keine Bedrohung gefunden wird, haben wir die Kategorie nicht in den Bericht aufgenommen.

Es wurden im Rahmen der Untersuchung insgesamt 9 Schwachstellen identifiziert, wobei vier (4) mit der Kritikalität *Hoch* eingestuft wurden. Neben den vier (4) hohen Schwachstellen konnten noch eine (1) Schwachstelle der Kategorie *Kritisch* gefunden werden.

Ein besonderes Risiko stellt die eine (1) Schwachstelle dar, welche als *Kritisch* eingestuft wurde. Bei dieser Sicherheitslücke handelt es sich um eine fehlerhafte Konfiguration im Logging der Android-App. Diese erlaubt die direkte Übernahme von Benutzerkonten und sollte daher mit höchster Priorität behoben werden. Da es sich hierbei aber nur um Fehlkonfiguration handelt, sollte diese mit wenig Aufwand behoben werden können.

Die identifizierten Schwachstellen mit der Kritikalität *Hoch* sollten zeitnah behoben werden, da diese den Zugriff auf sensible Daten erlauben, sowie eine Kompromitierung der gesamten Anwendung nicht ausgeschlossen werden kann. Die ungefilterte Verwendung von Benutzereingaben in Datenbankabfragen ist unsicher, da diese Schwachstellen das Vollständige auslesen aller Daten der Datenbank erlaubt.

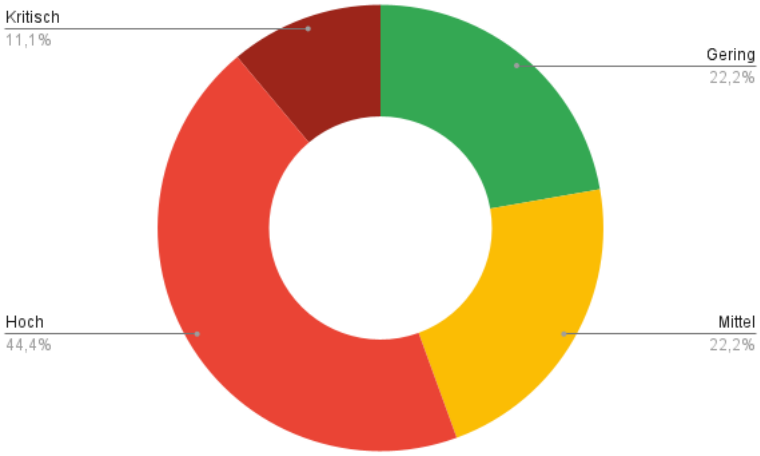


Abbildung 2.1: Verteilung Schwachstellen für die Franken Logistik GmbH

Zwei (2) der gefundenen Schwachstellen sind als *Mittel* zu bewerten. Diese Findings sollten in absehbarer Zeit geschlossen werden. Bei einer der Schwachstellen muss lediglich die eingesetzte Bibliothek zum Aufbau von SQL-Requests angepasst werden. Bei der anderen mittleren Schwachstelle handelt es sich um ein organisatorisches Finding. Es sollte zeitnah eine Dokumentation der API angefertigt werden, um so die Wartung der dieser zu erleichtern.

Alle mit der Kritikalität *Niedrig* eingestuften gefundenen Schwachstellen können ohne weitere Aufwände durch Setzen von jeweils gültigen Optionen sehr einfach beseitigt werden und haben voraussichtlich kaum Auswirkungen auf die Franken Logistik GmbH. Diese Schwachstellen können allesamt durch eine Anpassung der Konfiguration beseitigt werden.

Die Anwendung "Example" verfügt über ein unterdurchschnittliches Sicherheitsniveau.

Im Vergleich zu vorher von der AWARE7 durchgeführten Penetrationstests wurden nicht mehr Schwachstellen als sonst identifiziert. Dennoch sind die gefundenen Schwachstellen im Vergleich zu anderen bereits durchgeführten Penetrationstests deutlich kritischer.

Insgesamt konnten mehrere schwerwiegende Schwachstellen identifiziert werden, welche den weiteren sicheren Betrieb der mobilen- und Webanwendung behindern. Daher sollten die fünf identifizierten, als mindestens hoch klassifizierten Schwachstellen schnellstmöglich geschlossen werden. Davor kann ein sicherer Betrieb der Anwendung und Kundendaten nicht gewährleistet werden.

Eine technische Sicherheitsanalyse muss immer als Betrachtung zu einem Stichtag aufgefasst werden. Die zu diesem Tag bekannten Schwachstellen und Angriffstechniken werden genutzt. Aussagen über die Sicherheit eines Systems für die Zukunft lassen sich nicht treffen.

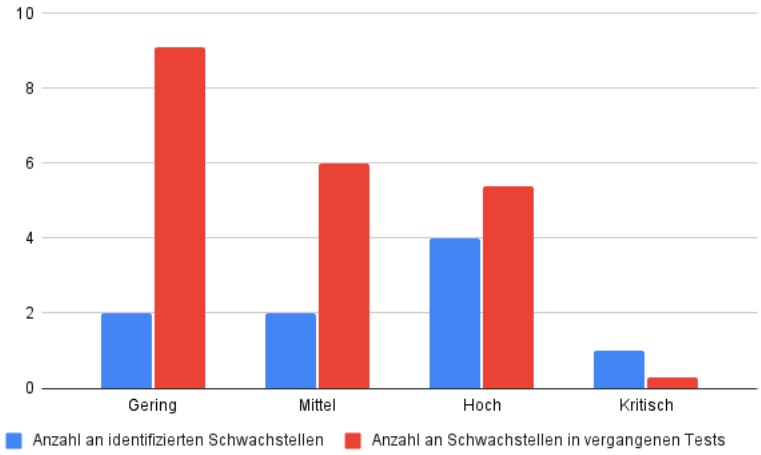


Abbildung 2.2: Verteilung Schwachstellen für die Franken Logistik GmbH im Vergleich

2.1.1. Mögliche Incident-Response Simulationen

Im Rahmen des Penetrationstests erhalten wir Einblicke in die Infrastruktur und können Incident Response Szenarien identifizieren, bei denen es hinsichtlich der allgemeinen Unternehmensresilienz sinnvoll sein kann, diese zu simulieren. Ein mögliches Szenario ist der Ausfall der API oder die Kompromittierung des Web Servers.

- Wie schnell kann in einem solchen Fall reagiert werden?
- Welche Prozesse werden intern angestoßen, welche extern?

Ein weiteres Szenario könnte sein, dass ein Angreifer die App dekompiliert und über gespeicherte Zugangsdaten zugriff auf interne Resource erhält.

- Welche Möglichkeiten hat der Angreifer dann?
- Wie reagiert das Management in einem solchen Fall? Welche Risiken bestehen für die Nutzer:innen der Franken Logistik GmbH?

Im Rahmen dieser Szenarien sollte auch die Dokumentation mit im Fokus stehen und überprüft werden.

- Gibt es geregelte Verantwortungen und Ansprechpartner für einen Notfall?
- Sind Notfallhandbücher aktuell und verfügbar?
- Gibt es Kontakte zu externen Dienstleistern die kurzfristig hinzugezogen werden können?

2.2. Technische Zusammenfassung

Alle Findings sind in nachfolgender tabellarischer Übersicht zusammengefasst. Dies ermöglicht einen Überblick über die Auffälligkeiten in der Reihenfolge der Kritikalität.

Risiko	Kap.	Beschreibung	Empfehlung	Seite
Kritisch	5.1	Die untersuchte mobile Anwendung schreibt sensible Login Daten in eine Logdatei, welche systemweit lesbar ist.	Die Daten sollten nicht systemweit lesbar gespeichert werden.	S. 21
Hoch	3.1	Die Webanwendung ist anfällig für eine Stored-XSS Schwachstelle.	Alle Eingaben von Benutzern müssen validiert werden.	S. 9
Hoch	3.2	Authentifizierte Nutzer können Benutzer mit höheren Rechten anlegen.	Das Rechtemanagement muss die Berechtigungen eines Nutzers überprüfen.	S. 12
Hoch	4.1	Durch eine unsichere Referenzierung von Objekten und Daten kann auf Objekte wie Profile oder Daten direkt zugriffen werden kann, wenn deren ID bekannt ist	Es sollte sichergestellt werden, dass ein Benutzer nur die Daten abrufen und ändern kann die zu seinem Profil gehören	S. 16
Hoch	4.2	Die API ist Anfällig für SQL-Injections. Dadurch kann gesamte Datenbank ausgelesen werden.	Sämtliche Variablen, die in einem SQL-Statement bearbeitet werden sollen müssen validiert werden.	S. 18
Mittel	5.2	Die App ist anfällig für lokale SQL-Injections.	Für den Benutzer sollte keine Möglichkeit bestehen, SQL-Statements manuell zu bearbeiten.	S. 23
Mittel	6.1	Die API-Schnittstellen sind nicht vollständig dokumentiert.	Der Dienstleister muss eine vollständige und korrekte Dokumentation liefern.	S. 25
Gering	4.3	Die Anwendung erlaubt beliebig viele Anfragen innerhalb kürzester Zeit.	Es sollte ein Ratelimiting eingeführt werden, dass die Anfragen in einem gewissen Zeitraum festlegt.	S. 20
Gering	3.3	Die gewählten HTTPS Einstellungen erlauben keine Forward Secrecy Eigenschaften.	Verwenden von vom BSI empfohlenen sicheren Verfahren.	S. 14

Tabelle 2.1: Technische Zusammenfassung aller Findings

3

Findings - www.example.com

Im folgenden Kapitel werden alle Schwachstellen des Servers *www.example.com* gelistet. Die Findings sind nach ihrer Kritikalität in absteigender Reihenfolge aufgelistet.

3.1. Stored Cross-Site-Scripting - Hoch

Exploitability Metrics		Impact Metrics	
Attack Vector (AV)	Network	Condentiality Impact (C)	High
Attack Complexity (AC)	Low	Integrity Impact (I)	Low
Privileges Required (PR)	High	Availability Impact (A)	None
User Interaction (UI)	Required		
Scope (S)	Unchanged		
CVSS Score: 8.5			

Benutzereingaben innerhalb eines Portals werden in vielen Bereichen genutzt, um dargestellte Informationen anzupassen. Das kann sich z.B. auf die Darstellung des Benutzernamens beziehen. Enthalten diese Benutzereingaben Script-Code, kann dieser durch den Browser unbeabsichtigt zur Ausführung gebracht werden.

Beschreibung der gefundenen Schwachstelle

Die Anwendung weist eine Cross-Site-Scripting (XSS) Schwachstelle auf. Innerhalb der Webseite und auch der API gibt es die Möglichkeit, eigenen Text hochzuladen. Vor der Veröffentlichung wird dieser in einer Vorschau angezeigt. Dabei ist es möglich, innerhalb der Textfelder HTML-Tags einzubinden.

Details zu der Schwachstelle

Die Webanwendung sowie die API sind anfällig für Stored Cross-Site-Scripting. Dies beruht hauptsächlich auf einer fehlenden Eingabevalidierung. An den folgenden Stellen konnten Daten eingebunden werden:

- https://api.example.com/api/v1/event
  Parameter: title

- https://api.example.com/api/v1/user/1-5698-76
  Parameter: param

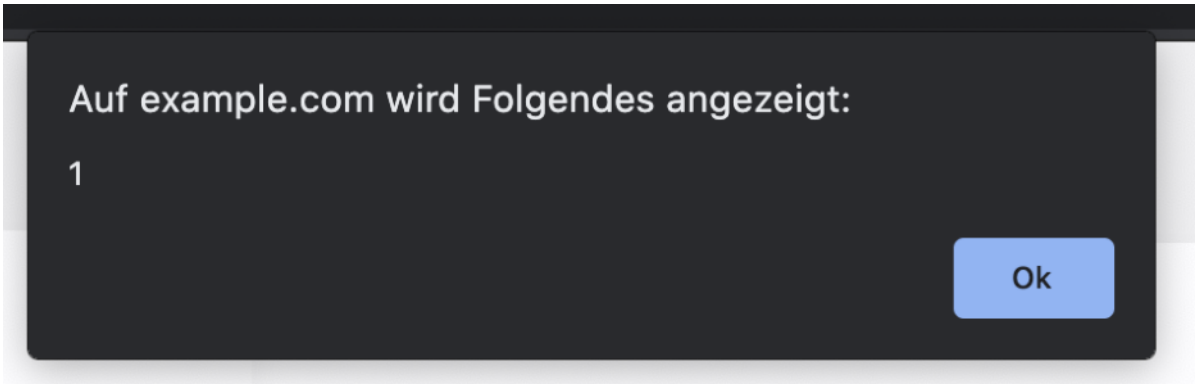


Abbildung 3.1: PoC der stored Cross-Site-Scripting Schwachstelle in der Webanwendung

```
curl -i -s -k -X $'POST' \
    -H $'Host: api.example.com' -H $'Authorization: Bearer
    ↪ yVJbnH7KaEKsu2FWJpNwe1OHrsA' -H $'Accept: */*' -H $'Sec-Fetch-Site:
    ↪ same-origin' -H $'Sec-Fetch-Mode: cors' -H $'Sec-Fetch-Dest: empty' -H
    ↪ $'Accept-Encoding: gzip, deflate' -H $'Accept-Language:
    ↪ de-DE,de;q=0.9,en-US;q=0.8,en;q=0.7' \
    --data-binary $'{\"param\": \"<script>alert(1)</script>\"}' \
    $'https://www.google.de/api/v1/user/1-5698-76'
```

Abbildung 3.2: Beispiel für einen curl Aufruf der XSS

Potenzielles Risiko

Durch die Ausnutzung der Schwachstellen besteht für einen Angreifer die Möglichkeit, eigene Inhalte dauerhaft in die Anwendung einzubinden. Davon ausgehend gibt es unter anderem die Bedrohungsszenarien:

- Verteilung von Schadcode: Nutzer können auf Seiten mit Schadcode umgeleitet werden.

- Diebstahl von Zugangsdaten: Die Struktur der Webseite kann so geändert werden, dass auf Zugangsdaten von anderen Nutzer zugegriffen werden kann (Cookie, Passwort).

Empfohlene Gegenmaßnahmen

Um die Anwendung vor Cross-Site-Scripting zu schützen, müssen alle Benutzereingaben gefiltert werden. Es ist wichtig sicherzustellen, dass Benutzerdaten- und Befehle getrennt behandelt werden. Zum Filtern der Benutzereingaben wird empfohlen, eine API (wie z.B. OWASP ESAPI) zu verwenden. Die vorgestellte API bietet verschiedene Schnittstellen für verschiedene Arten von Eingaben wie z.B. Strings oder ganze Zahlen. Ebenfalls gefiltert werden alle HTML-Zeichen und Funktionen von z.B. PHP.

3.2. Session Management Schema - Hoch

Exploitability Metrics		Impact Metrics	
Attack Vector (AV)	Network	Condentiality Impact (C)	High
Attack Complexity (AC)	Low	Integrity Impact (I)	Low
Privileges Required (PR)	Low	Availability Impact (A)	None
User Interaction (UI)	None		
Scope (S)	Unchanged		
CVSS Score: 7.1			

Eine der Kernkomponenten jeder webbasierten Anwendung ist der Mechanismus, mit dem sie den Status eines Benutzers, der mit ihr interagiert, kontrolliert und aufrechterhält. Session Management bezeichnet verschiedene Mechanismen zur Speicherung und Validierung von Anmeldedaten für einen bestimmten Zeitraum.

Beschreibung der gefundenen Schwachstelle

Session-IDs sind (nach Benutzername und Passwort) das zentrale Authentisierungsmerkmal bei der Benutzung eines Web-Portals. Entsprechend sicher muss die Initiierung und der Transport von Session-IDs realisiert werden. Im vorliegenden Test wird daher das Session-Management des Portals überprüft.

Details zu der Schwachstelle

Die genutzte Session-ID `hid_s` sind so zufällig, dass nicht davon ausgegangen werden kann, dass ein Angreifer einen Cookie erfolgreich erraten kann.

Die statistische Qualität der Session Cookies ist: exzellent: `hid_s` hat einen exzellenten Entropie-Wert (133 Bit bei 11000 Tokens und 1% Signifikanz)

Benutzern werden innerhalb des Portals bei einem Login-Vorgang Rollen und zugehörige Rechte zugewiesen. Es wird daher geprüft, ob ein Benutzer die eigene Rolle verlassen und Befehle für andere Benutzer (mit einer anderen, höheren Rolle) ausführen kann.

Es wurde festgestellt, dass es für einen authentifizierten Benutzer möglich ist, mehr Privilegien zu erhalten und Root-Adminstrator zu werden. Ein authentifizierter Benutzer ist in der Lage, auf das folgende API-Backend und die folgenden Funktionen zuzugreifen:

- `https://api.example.com/api/v1/user`

Der Benutzer ist in der Lage, eine manipulierte HTTP-POST-Anfrage mit einem JSON-Body zu erstellen, der zur Generierung eines neuen Benutzers verwendet werden kann. Wenn

ein neuer Benutzer erzeugt wird, enthält dieses JSON-Dokument einen spezifischen „Role“ Parameter. Es ist möglich, dass ein authentifizierter Benutzer mit der Rolle „User“ oder „Manager“ einen neuen Benutzer mit der Rolle „Root“ erstellt.

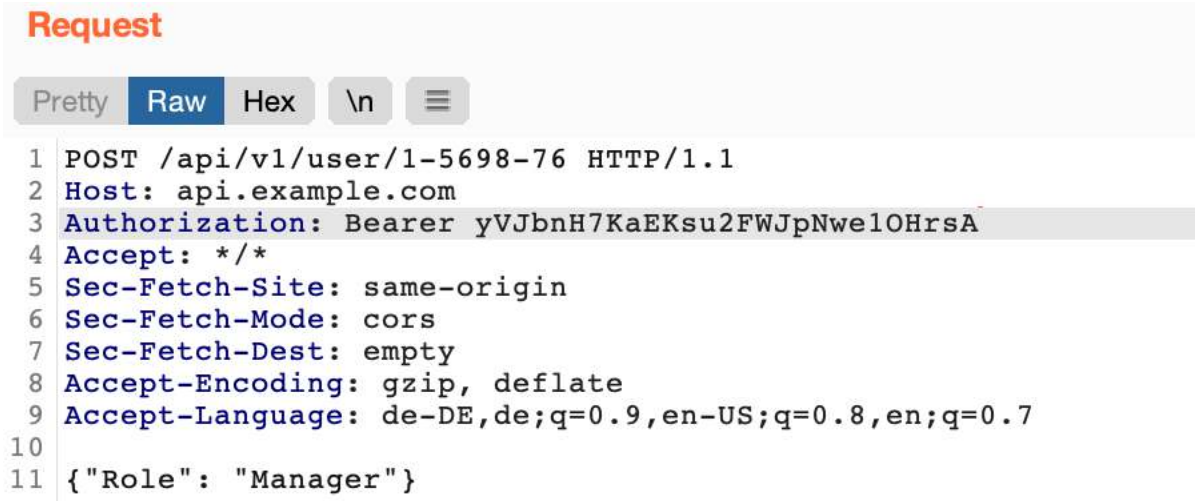


Abbildung 3.3: Ausnutzung von Fehlern im Session Management Schema

Potenzielles Risiko

Eine solche Privilege Escalation Schwachstelle ermöglicht es einem Benutzer mit geringen Privilegien Zugang zu Funktionen und sensiblen Daten zu erhalten, zu deren Abfrage und Interaktion er nicht berechtigt ist.

Empfohlene Gegenmaßnahmen

Die Logik der Prozesse muss überprüft werden. Das Rechtemanagement muss angepasst werden, so dass sichergestellt wird, dass Benutzer:innen die Rechte nicht erweitern können.

3.3. Unzureichende HTTPS-Konfiguration - Gering

Exploitability Metrics		Impact Metrics	
Attack Vector (AV)	Adjacent	Condentiality Impact (C)	None
Attack Complexity (AC)	Low	Integrity Impact (I)	Low
Privileges Required (PR)	Low	Availability Impact (A)	None
User Interaction (UI)	Required		
Scope (S)	Unchanged		
CVSS Score: 3.5			

Das Hypertext Transfer Protocol Secure (HTTPS) ist ein Kommunikationsprotokoll mit dem Daten abhörsicher übertragen werden können.

Beschreibung der gefundenen Schwachstelle

Die gewählten SSL Einstellungen stellen keine Forward Secrecy sicher.

Details zu der Schwachstelle

Forward Secrecy soll sicher stellen, dass auch zukünftig die verschlüsselten Daten nicht entschlüsselt werden können. Die Kompromittierung eines Schlüssels führt nicht zur Kompromittierung vorheriger Schlüssel. Diese Eigenschaft unterstützen nur bestimmt Algorithmen.

Abbildung 3.4: Bewertung von eingesetzten TLS-Algorithmen

Potenzielles Risiko

Ausgehend von einem Sitzungsschlüssel werden weitere Schlüssel abgeleitet. Wird der Sitzungsschlüssel kompromittiert können daraus abgeleitete Schlüssel berechnet und Nachrichten entschlüsselt werden.

Empfohlene Gegenmaßnahmen

Es sollten TLS-Algorithmen zur Verschlüsselung verwendet werden, die Forward Secrecy ermöglichen. Aktuell empfiehlt sich der Einsatz der folgenden Verfahren:

- TLS_ECDHE_PSK_WITH_AES_128_CCM_8_SHA256
- TLS_AES_128_GCM_SHA256

4

Findings - api.example.com

Im folgenden Kapitel werden alle Schwachstellen des API-Servers *api.example.com* gelistet. Die Findings sind nach ihrer Kritikalität in absteigender Reihenfolge aufgelistet.

4.1. Insecure Direct Object Referencing - Hoch

Exploitability Metrics		Impact Metrics	
Attack Vector (AV)	Network	Condentiality Impact (C)	High
Attack Complexity (AC)	Low	Integrity Impact (I)	High
Privileges Required (PR)	Low	Availability Impact (A)	None
User Interaction (UI)	None		
Scope (S)	Unchanged		
CVSS Score: 8.1			

Beschreibung der gefundenen Schwachstelle

Es konnte eine unsichere Referenzierung von Objekten und Daten gefunden werden. Bei Insecure Direct Object Referencing handelt es sich um eine Schwachstelle bei der auf Objekte wie Profile oder Daten direkt zugriffen werden kann, wenn deren ID bekannt ist.

Details zu der Schwachstelle

Es konnte festgestellt werden, dass personenbezogene Daten von anderen Benutzern einsehbar und änderbar sind. Um auf die Daten von anderen Benutzern zuzugreifen muss muss nur eine gültige numerische User-ID, wie die folgende, erraten werden.

- `https://api.example.com/api/v1/user/1-5698-76/`

Potenzielles Risiko

Ein Angreifer kann versuchen durch das Erraten von User-IDs sensible personenbezogene Daten einzusehen und zu ändern.

Request

Pretty Raw Hex \n

```
1 GET /api/v1/user/1-5698-76/ HTTP/1.1
2 Host: api.example.com
3 Authorization: Bearer yVJbnH7KaEKsu2FWJpNwe1OHrsA
4 Accept: */*
5 Sec-Fetch-Site: same-origin
6 Sec-Fetch-Mode: cors
7 Sec-Fetch-Dest: empty
8 Accept-Encoding: gzip, deflate
9 Accept-Language: de-DE,de;q=0.9,en-US;q=0.8,en;q=0.7
10
11
```

Abbildung 4.1: Insecure Direct Object Referencing im User-Endpunkt

Empfohlene Gegenmaßnahmen

Es sollte sichergestellt werden, dass ein Benutzer nur die Daten abrufen und ändern kann die zu seinem Profil gehören.

4.2. Header-based Blind SQL-Injections - Hoch

Exploitability Metrics		Impact Metrics	
Attack Vector (AV)	Network	Condentiality Impact (C)	High
Attack Complexity (AC)	Low	Integrity Impact (I)	None
Privileges Required (PR)	Low	Availability Impact (A)	None
User Interaction (UI)	None		
Scope (S)	Changed		
CVSS Score: 7.7			

Beschreibung der gefundenen Schwachstele

Die API auf dem System `api.example.com` weißt mehrere SQL-Injections auf. Durch SQL-Injections kann die gesamte Datenbank ausgelesen werden und sensible Informationen gestohlen werden.

Details zu der Schwachstelle

In zwei Funktionen der API konnte eine Blind SQL-Injection gefunden werden, die das Auslesen der gesamten Datenbank möglich macht. Die Funktionen *validateToken* und *RenewToken* verarbeiten die HTTP-Header Variable *token*, ohne diese zu validieren und somit SQL-Injections zu verhindern.

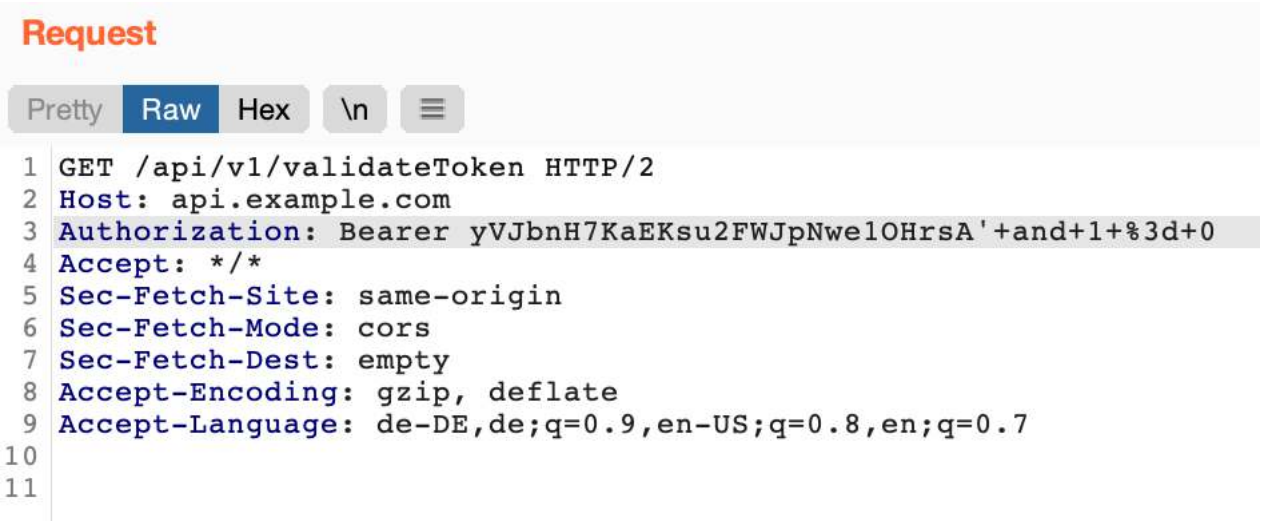


Abbildung 4.2: Blind SQL-Injection in der validateToken API-Anfrage

Potenzielles Risiko

Durch die Ausnutzung der Schwachstellen bestehen für einen Angreifer die Möglichkeit die Inhalte der Datenbank unbeschränkt einzusehen.

Empfohlene Gegenmaßnahmen

Es sollte sichergestellt werden, dass sämtliche Benutzereingaben gefiltert werden. Daher sollten auch HTTP-Header, bevor diese verarbeitet werden, validiert werden. Insgesamt empfiehlt sich der Einsatz von Prepared-Statements zur konsequenten Vermeidung von SQL-Injections.

4.3. Fehlendes Ratelimiting - Gering

Exploitability Metrics		Impact Metrics	
Attack Vector (AV)	Network	Condentiality Impact (C)	None
Attack Complexity (AC)	High	Integrity Impact (I)	None
Privileges Required (PR)	None	Availability Impact (A)	Low
User Interaction (UI)	None		
Scope (S)	Unchanged		
CVSS Score: 3.7			

Beschreibung der gefundenen Schwachstelle

Die Anwendung erlaubt beliebig viele Anfragen innerhalb kürzester Zeit und bearbeitet diese. Dadurch sind verschiedene Angriffe möglich, die durch ein Ratelimiting vermieden werden können.

Details zu der Schwachstelle

Während der gesamten Testphase konnte kein Ratelimiting erkannt werden, bzw. wurden die anfragenden Systeme blockiert.

Potenzielles Risiko

Durch ein fehlendes Ratelimiting sind Angriffe wir Brute-Forcing oder auch Denial of Service möglich.

Empfohlene Gegenmaßnahmen

Es sollte ein Ratelimiting eingeführt werden, dass die Anfragen in einem gewissen Zeitraum festlegt. Ab einer bestimmten Anzahl von Request pro wird es sich um automatische und maschinelle Anfragen handeln. Ein guter Richtwert, um mit dem Ratelimiting automatisierte Anfragen zu blockieren, liegt bei ca. 50 Anfragen pro Minute.

5

Findings - com.example.app

Im folgenden Kapitel werden alle Schwachstellen der mobilen Android-Anwendung *com.example.app* gelistet. Die Findings sind nach ihrer Kritikalität in absteigender Reihenfolge aufgelistet.

5.1. Preisgabe sensibler Log-Daten - Kritisch

Exploitability Metrics		Impact Metrics	
Attack Vector (AV)	Local	Condentiality Impact (C)	High
Attack Complexity (AC)	Low	Integrity Impact (I)	High
Privileges Required (PR)	None	Availability Impact (A)	None
User Interaction (UI)	None		
Scope (S)	Changed		
CVSS Score: 9.0			

Beschreibung der gefundenen Schwachstelle

Die untersuchte, mobile Anwendung schreibt sensible Daten in eine Logdatei, welche systemweit lesbar ist.

Details zu der Schwachstelle

Die Anwendung *com.example.app* loggt Daten beim Login mit. Diese Daten befinden sich im folgenden Ordner, welcher von allen Apps mit Zugriff auf die SD-Karte gelesen werden kann.

- /storage/emulated/0/com.example.app/

Dadurch können andere mobile Apps auf dem Smartphone die genutzten Login-Daten des Benutzers einsehen.

```
[FILESYSTEM] Logging into /storage/emulated/0/com.example.app/log.txt
...
[REQUEST] POST /api/v1/getToken {"user":"aware7","password":"Zjhg12*"} - 200 OK
[REQUEST] GET /api/v1/validateToken - 200 OK
...
```

Abbildung 5.1: Zugangsdaten im Systemweiten Logging

Potenzielles Risiko

Eine andere Anwendung auf dem mobilen Gerät wäre in der Lage, die Logdatei mitzulesen und so die dort gespeicherten Daten einzusehen. Da die dort gespeicherten Informationen, den Username und das Passwort enthalten, kann somit das Konto des Benutzers übernommen werden.

Empfohlene Gegenmaßnahmen

Es sollte evaluiert werden, ob das Loggen dieser Daten benötigt wird. Sollte dies nicht der Fall sein, sollte der Umfang des Loggings reduziert werden.

5.2. Raw SQL-Queries - Mittel

Exploitability Metrics		Impact Metrics	
Attack Vector (AV)	Local	Condentiality Impact (C)	Low
Attack Complexity (AC)	Low	Integrity Impact (I)	None
Privileges Required (PR)	Low	Availability Impact (A)	None
User Interaction (UI)	None		
Scope (S)	Unchanged		
CVSS Score: 4.0			

Beschreibung der gefundenen Schwachstelle

Die Applikation verwendet eine SQLite-Datenbank und führt darauf native SQL-Statements aus, welche im Quellcode ersichtlich sind.

Details zu der Schwachstelle

Die mobile Android Anwendung speichert Daten des Benutzers in einer lokalen SQLite-Datenbank. Die Abfragen an diese Datenbank können über die *Suchmaske* für Offline-Daten manipuliert werden. Dadurch ist die Anwendung anfällig für eine lokale SQL-Injection.

```
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class c extends SQLiteOpenHelper {
    private static final String f1626d = ("CREATE TABLE IF NOT EXISTS " + h17bs
        + " (" + "id" + " INTEGER PRIMARY KEY AUTOINCREMENT, " + "type" + " TEXT,
        " + "time" + " LONG, " + "result" + " TEXT )");


    public void onCreate(SQLiteDatabase sQLiteDatabase) {
        sQLiteDatabase.beginTransaction();
        try {
            sQLiteDatabase.execSQL(f1626d);
            [...]
```

Abbildung 5.2: SQL-Statments

Potenzielles Risiko

SQL-Statements sollten nicht direkt vom Benutzer vorgegeben werden können. Es besteht das Risiko, dass sensible Informationen eingesehen oder der Datenbestand manipuliert bzw. gelöscht werden kann.

Empfohlene Gegenmaßnahmen

Für den Benutzer sollte keine Möglichkeit bestehen, SQL-Statements manuell zu bearbeiten. Alle sensiblen Informationen der Datenbank sollten verschlüsselt werden und lediglich systemintern verarbeitet werden.

6

Findings - Organisatorisch

Im folgenden Kapitel werden organisatorische beziehungsweise sonstige Findings gelistet. Die Findings sind nach ihrer Kritikalität in absteigender Reihenfolge aufgelistet.

6.1. Fehlerhafte Dokumentation der API-Schnittstellen - Mittel

Beschreibung der gefundenen Schwachstelle

Seitens des Kunden konnte keine vollständige Schnittstellenbeschreibung der API geliefert werden.

Details zu der Schwachstelle

Die API-Dokumentation war nicht vollumfänglich nutzbar und sollte durch den Kunden überarbeitet werden. Insbesondere folgende Punkte sind aufgefallen:

- Fehlende Beispielanfragen
- Fehlende Beschreibung der Inhalte der versendeten Parameter
- Fehlende Beschreibung der Verschlüsselung
- unvollständige Dokumentation der bereitgestellten Endpunkte

Potenzielles Risiko

Eine detaillierte Dokumentation zu den API-Schnittstellen kann im Notfall wichtig sein, um schnell einen Überblick der Systeme erhalten zu können. Darüber hinaus ist eine detaillierte Dokumentation sinnvoll, damit Dienstleister, einen Überblick über die Systeme erhalten können.

Empfohlene Gegenmaßnahmen

Es sollte eine korrekte und umfassende Beschreibung aller Funktionen der API existieren.

7

Portscans

Die folgenden Tabellen zeigen die Ergebnisse der Port-Scans der beauftragten Systeme. Ein Port-Scan identifiziert Dienste, welche auf einem Server oder System nach außen hin angeboten werden. Ein offener Port stellt per se kein Sicherheitsrisiko dar, solange der installierte Dienst sicher betrieben wird. Es ist empfehlenswert zu evaluieren, ob die Dienste auf den hier gelisteten Systemen verwendet werden oder nicht. Nicht verwendete Dienste sollten wenn möglich immer deaktiviert werden, damit diese nicht zum Sicherheitsrisiko werden können.

7.1. api.example.com

Port	Status	Dienst	Version
80/tcp	Offen	HTTP	nginx
443/tcp	Offen	SSL/HTTP	nginx
53/tcp	Gefiltert	DNS	

Tabelle 7.1: Port Scan von api.example.com

7.2. example.example.com

Port	Status	Dienst	Version
22/tcp	Offen	SSH	OpenSSH 7.7.2 (Ubuntu)
80/tcp	Offen	HTTP	nginx
443/tcp	Offen	SSL/HTTP	nginx
8443/tcp	Offen	SSL/HTTP	httpd
78443/tcp	Gefiltert	-	

Tabelle 7.2: Port Scan von example.example.com

7.3. www.example.com

Port	Status	Dienst	Version
21/tcp	Offen	FTP	Pure-FTPd 1.0.48
22/tcp	Offen	SSH	OpenSSH 7.7.2 (Ubuntu)
80/tcp	Offen	HTTP	nginx
443/tcp	Offen	SSL/HTTP	nginx

Tabelle 7.3: Port Scan von www.example.com

Tabellenverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

BSI Bundesamt für Sicherheit in der Informationstechnik

EC Elliptic Cruve

TLS Transport Layer Security

SSL Secure Sockets Layer

HTML5 Hypertext Markup Language 5

CORS Cross-Origin Resource Sharing

HTTP Hypertext Transfer Protocol

HTTPS Secure Hypertext Transfer Protocol

XSS Cross-Site-Scripting

API Application Programming Interface

DOS Denial-of-Service

CSP Content-Security-Policy

MIME Multipurpose Internet Mail Extensions

HSTS HTTP Strict Transport Security

MitM Man-in-the-Middle

JSON JavaScript Object Notation

XML Extensible Markup Language

CSS Cascading Style Sheets

DiGAV Digitale-Gesundheitsanwendungen-Verordnung

ISMS Information Security Management System

AD Active Directory

VPN Virtual Private Network

SQL Structured Query Language

VoIP Voice over IP

LDAP Lightweight Directory Access Protocol

NAC Network Access Control

IOT Internet of Things

NetBIOS Network Basic Input Output System

SMB Server Message Block

IPv6 Internet Protocol Version 6

MDNS Multicast DNS

SMTP Simple Mail Transfer Protocol

RPC Remote Procedure Call

OWA Outlook on the web

MDM Mobile Device Management

SSH Secure Shell