Projekt Luna

Pentest Webanwendung

Ergebnisbericht

MindBytes GmbH | Probststraße 15 | 70567 Stuttgart

+49 711 20709567 | hallo@mind-bytes.de

Geschäftsführung: Christian Stehle, Nina Wagner, Simon Holl
HRB 790784 | Amtsgericht Stuttgart

Version 1.0
Vertraulich
Kontakt: hallo@mind-bytes.de

Musterfirma GmbH

Inhalt

1 Management Summary

Testgegenstand: Webanwendung Example **Handlungsbedarf:** Dringend

Gesamtrisiko

- Mehrere Schwachstellen erlauben den Zugriff auf die Daten anderer Kunden. Unter anderem sind auch personenbezogene Daten betroffen. Das kann datenschutzrechtliche Konsequenzen haben und Rufschäden nach sich ziehen.
- Die Sicherheit von Benutzerkonten ist durch unterschiedliche Sachverhalte gefährdet. Angreifer können so die volle Kontrolle über Konten übernehmen und Daten einsehen, ändern oder löschen.
- Die Monitoring-Endpunkte des Webservers offenbaren sensible technische Informationen, die für normale Benutzer der Anwendung nicht relevant sind. Beispielsweise werden die Zugangsdaten der Datenbank oder die Sitzungs-Cookies anderer Benutzer preisgegeben. Die Informationen sind für das weitere Vorgehen von Angreifern nützlich.
- Weitere konfigurative Maßnahmen können die Sicherheit der Anwendung weiter verbessern und die Ausnutzung von Schwachstellen erschweren.

Gesamtrisiko im Vergleich zu anderen Unternehmen[1]**:** Schlechter als Durchschnitt

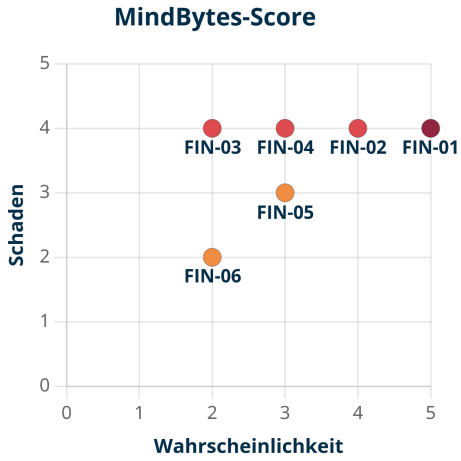

Abbildung 2 - Verteilung nach Schaden und Wahrscheinlichkeit

Abbildung 3 - Verteilung nach Risiko

[1] Dies ist eine relative Einschätzung und lässt keine Rückschlüsse auf die Gefährdungslage zu.

1.1 Handlungsempfehlung

Die Einschätzung zur Behebung basiert auf unserer Erfahrung und sollte intern validiert werden. In der Regel resultieren erfolgreiche Angriffe aus der Verkettung von mehreren Schwachstellen, weshalb wir eine Behebung aller Findings empfehlen. Beim Umsetzen von Maßnahmen ist es wichtig, Schwachstellen nicht als Einzelfall zu betrachten, sondern an der Ursache zu arbeiten, um ähnlichen Schwachstellen in der Zukunft vorzubeugen.

Maßnahme	Behebung	Hinweise zur Behebung	Findings
Actuator entfernen ⚡	⌚ Dringend ⌛ Stunden 💰 Nein	▪ Quick-Win: mit geringem Aufwand kann die Sicherheit deutlich erhöht werden	3.4 FIN-04: Spring Boot Actuator aktiv
Benutzereingaben validieren	⌚ Dringend ⌛ Tage 💰 Nein	▪ Benutzereingaben sollten generell als bösartig angesehen und vor der Verarbeitung geprüft werden	3.1 FIN-01: Manipulieren von Datenbankabfragen – SQL-Injection 3.3 FIN-03: Einschleusen bösartiger Inhalte in Website-Cache
Übernahme von Benutzerkonten vorbeugen	⌚ Dringend ⌛ Tage 💰 Nein	▪ Kombination der Findings ermöglicht die vollständige Übernahme anderer Benutzerkonten	3.2 FIN-02: Impersonisieren anderer Benutzer 3.5 FIN-05: Ausführung von sensiblen Aktionen ohne Identitätsüberprüfung
Webserver härten	⌚ Mittelfristig ⌛ Stunden 💰 Nein	▪ Sicherheit der Anwendung weiter verbessern	3.6 FIN-06: Fehlende HTTP-Security-Header

⌚ Priorität: dringend / mittelfristig / langfristig | ⌛ Geschätzte Behebungsdauer je Finding: Stunden / Tage / Wochen | 💰 Entstehen Kosten: nein / vermutlich (nicht) / ja

2 Technical Summary

2.1 Findings-Tabelle

Finding	CVSS-Score (v3.1)	MindBytes-Score Schaden	MindBytes-Score Wahrscheinlichkeit
3.1 FIN-01: Manipulieren von Datenbankabfragen – SQL-Injection 💡 Parametrisierte Abfragen für Datenbankabfragen verwenden	9.3 (Critical)	🔥🔥🔥🔥🔥	🎲🎲🎲🎲🎲
3.2 FIN-02: Impersonisieren anderer Benutzer 💡 Signaturprüfung von JWT implementieren	8.1 (High)	🔥🔥🔥🔥🔥	🎲🎲🎲🎲🎲
3.3 FIN-03: Einschleusen bösartiger Inhalte in Website-Cache 💡 Website-Inhalte und insbesondere Cache ohne Benutzereingaben erstellen	7.5 (High)	🔥🔥🔥🔥🔥	🎲🎲🎲🎲🎲
3.4 FIN-04: Spring Boot Actuator aktiv 💡 Spring Boot Actuator deaktivieren oder Funktionen einschränken	7.5 (High)	🔥🔥🔥🔥🔥	🎲🎲🎲🎲🎲
3.5 FIN-05: Ausführung von sensiblen Aktionen ohne Identitätsüberprüfung 💡 Passwortabfrage vor sensiblen Aktionen	5.6 (Medium)	🔥🔥🔥🔥🔥	🎲🎲🎲🎲🎲
3.6 FIN-06: Fehlende HTTP-Security-Header 💡 Konfigurieren von HTTP-Security-Headern	4.2 (Medium)	🔥🔥🔥🔥🔥	🎲🎲🎲🎲🎲

Details zu den einzelnen Findings sind im Kapitel 3 Findings beschrieben. Diesem Bericht liegen folgende Dateien bei:

📄 Grafische Auswertungen, tabellarische Übersicht der Findings und Asset-Liste mit Zuordnung, welches Asset von welchem Finding betroffen ist: Projekt-Luna-Übersicht.xslx

🔎 Technische Informationen, die an relevanten Stellen in den Findings referenziert werden:
Projekt-Luna-Technische-Infos.xlsx

2.2 Weiteres Vorgehen

1. Nachbereitung (vgl. Abschnitt 2.5 Nachbereitung)
2. Sichten und Nachvollziehen der Ergebnisse aus diesem Bericht, Klären von Fragen in der Abschlussbesprechung
3. Planung und Priorisierung von Behebungsmaßnahmen, z. B. mit der vorbereiteten Tabelle im Sheet „Findings-Übersicht" der angehängten Datei 📄
4. Umsetzung und Nachverfolgung von Behebungsmaßnahmen
5. Empfehlenswerte nächste Tests:
    - Retest der Ergebnisse zur Prüfung der Effektivität der getroffenen Behebungsmaßnahmen (geschätztes Budget: 2.000–5.000 €)
    - Physical Red Teaming zur Prüfung, wie leicht Unbefugte in Firmengebäude/Produktionshallen eindringen können (geschätztes Budget: 10.000–15.000 €)
    - Regelmäßige Wiederholung dieses Pentests, um Änderungen und evtl. neue Angriffstechniken zu prüfen

2.3 Ausgangspunkt im Projekt

Bereitgestellte Informationen[2]	Test-Umfang	Vorgehensweise	Ausgangspunkt[3]
keine (Black-Box)	vollständig	verdeckt (Red Teaming)	von außen
einige (Grey-Box)	**begrenzt**	**offensichtlich (Pentest)**	**von innen**
vollumfänglich (White-Box)	fokussiert		

[2] Details siehe Abschnitt 4.6 Bereitgestellte Informationen

[3] Details siehe Abschnitt 4.4 Zugriffsweg und 4.5 Bereitgestellte Benutzerkonten

2.4 Einschränkungen im Projekt

Es gab keine Faktoren, die die Durchführung des Projekts beeinträchtigten.

2.5 Nachbereitung

1. Erstellte Ausnahmen in vorhandenen Schutzsystemen löschen, sofern kein Retest oder Folgetest geplant ist
2. Bereitgestellte Zugänge (siehe Abschnitt 4.5 Bereitgestellte Benutzerkonten) deaktivieren, sofern ein Retest oder Folgetest geplant ist, andernfalls löschen
3. In Anwendung angelegte Firma MINDBYTES löschen

3 Findings

3.1 FIN-01: Manipulieren von Datenbankabfragen – SQL-Injection

Betroffen: CVSS v3.1: 9.3 (Critical)

- https://example.com/

3.1.1 Übersicht

Die Anwendung verwendet Benutzereingaben zur Erstellung von SQL-Abfragen. So können manipulierte Anfragen an die Datenbank gestellt und dadurch auf weitere gespeicherte Information zugegriffen werden.

Mögliche Folgen einer erfolgreichen Ausnutzung 🔥🔥🔥🔥🔥

- Ausführen von eigenen SQL-Abfragen
- Kann generell für unterschiedliche Aktionen genutzt werden:
  - Auslesen von weiteren gespeicherten sensiblen Daten
  - Provozieren von gültigen Datenbankabfragen ohne gültige Eingabewerte für das korrekte Ausführen von Anwendungsflows wie Logins
  - Verändern oder Löschen von Daten
  - Überlasten der Datenbank durch ressourcenhungrige Abfragen
  - Missbrauch von Datenbankfunktionen, beispielsweise für den Zugriff auf Dateien, das Ausführen von Betriebssystembefehlen oder das Stellen von Netzwerkanfragen an interne Systeme

Beispiele für Voraussetzungen für eine Ausnutzung

- Schwachstelle ließ sich ohne Authentifizierung ausnutzen
- Tiefergehendes Wissen zu SQL und der Schwachstelle ist vorteilhaft
- Es existieren Tools, die bei der Ausnutzung unterstützen

3.1.2 Empfehlung

- Prepared Statements mit parametrisierten Abfragen verwenden:
  - Damit können die Abfrage und die nötigen Benutzereingaben getrennt an die Datenbank übergeben werden
  - Die Datenbank behandelt Benutzereingaben korrekt, sodass keine Manipulation der Abfrage möglich ist
  - Eine Übersicht, wie sich das in den unterschiedlichen Programmiersprachen umsetzen lässt, bietet dieser Blogartkel
- Nach weiteren Stellen innerhalb der Anwendung suchen, die ebenfalls betroffen sein könnten

3.1.3 Technische Details

Die Webanwendung verwendet an mehreren Stellen Benutzereingaben, um Datenbank-Abfragen zu erstellen. Der Quellcode wurde mit dem Tool semgrep automatisiert auf diese Schwachstelle untersucht. Im Anhang *semgrep-sql.txt* sind die Ergebnisse dieses Tools und damit weitere potenziell anfällige Stellen aufgeführt. Da es sich um eine große Anzahl potenziell verwundbarer Stellen handelt, wurden diese nicht alle manuell überprüft.

Nachfolgend wird eine Stelle beschrieben, an der nach Umgehen des implementierten Filters manipulierte Abfragen erstellt und so Benutzerinformationen ausgelesen werden können. Die nebenstehende Abbildung zeigt die verwundbare Stelle mit implementiertem Filter in der Datei */includes/example_db.php*.

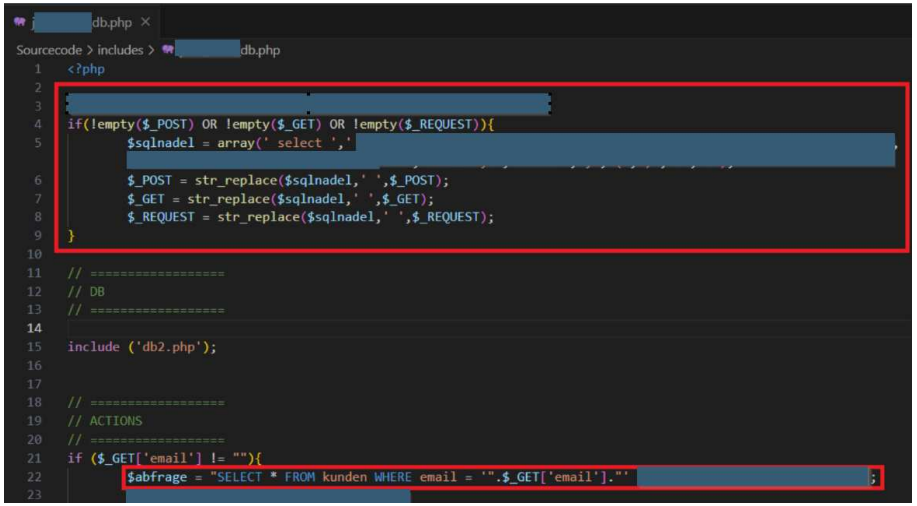

Abbildung 4 - Verwundbarer Quellcode

Beim Aufruf der URL https://example.com/includes/example_db.php?email=1\&hash=%20OR%201=1%20LIMIT%2010%20--%20 nutzt die Anwendung sowohl den Parameter *email* als auch *hash*, um eine Datenbank-Abfrage zu erstellen.

Die Eingaben werden zunächst gefiltert, um mögliche Manipulationen der Abfrage zu verhindern. Dieser Filter kann jedoch umgangen werden. Beim Aufruf der obigen URL wird folgende SQL-Abfrage konstruiert:

```
SELECT * FROM kunden WHERE email = '1\' AND hash = ' OR 1=1 LIMIT 10 -- '
```

Der eingefügte Backslash (`\`) aus dem Parameter *email* macht das nachfolgende Anführungszeichen unwirksam. Der Parameter *hash* wird dann dazu genutzt, eine immer gültige Bedingung (`1=1`) einzuführen und die Anzahl der Ergebnisse auf 10 Zeilen zu limitieren (`LIMIT 10`). Außerdem wird der Rest der vordefinierten Abfrage auskommentiert (`--`).

Außerdem kann der Filter umgangen werden, indem die SQL-Schlüsselwörter in Großbuchstaben geschrieben werden. Bei der Ausführung wird die Groß- und Kleinschreibung von der Datenbank nicht beachtet, bei dem implementierten Filter jedoch schon. Eine Ausnutzung ohne Wissen des implementierten Filters ist deutlich schwerer, jedoch nicht unmöglich. Es wird ein sehr zielgerichtetes Vorgehen benötigt, um zum Erfolg zu kommen.

Die Abbildung zeigt den Aufruf der genannten URL im Browser.

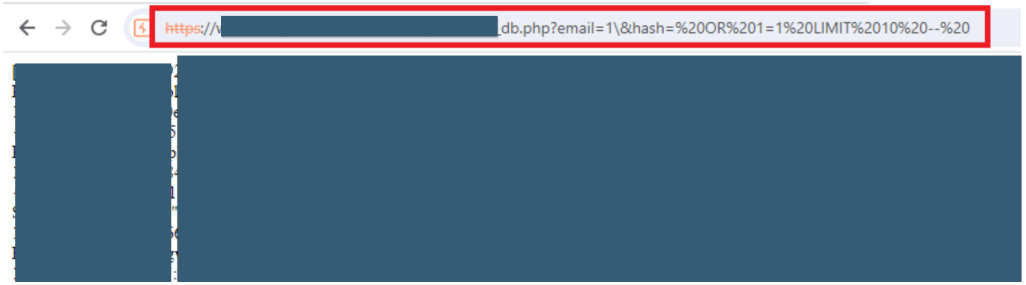

Abbildung 5 - Auswirkungen der manipulierten Datenbank-Abfrage

> Da sich einige mögliche verwundbare Stellen in SQL-Abfrage mit dem Keyword *UPDATE* befinden, gehen wir davon aus, dass auch Daten böswillig manipuliert werden könnten. Um den Betrieb nicht zu gefährden, überprüften wir dies nicht.

3.2 FIN-02: Impersonisieren anderer Benutzer

Betroffen: CVSS v3.1: 8.1 (High)

- https://example.com/

3.2.1 Übersicht

Durch eine Schwäche in der Validierung von Zugriffstokens können Benutzer beliebige andere Benutzerkonten übernehmen und darüber auf Daten anderer Organisation zugreifen.

Mögliche Folgen einer erfolgreichen Ausnutzung

- Übernahme anderer Benutzerkonten
- Ermöglicht insbesondere Zugriff auf Daten anderer Organisationen

Beispiele für Voraussetzungen für eine Ausnutzung

- Kenntnis eines gültigen JSON-Web-Tokens für einen beliebigen Benutzer, bspw. durch Zugriff auf die öffentlich verfügbare Demo-Umgebung
- Rudimentäre Kenntnisse über die Funktionsweise von JSON-Web-Tokens (JWT)
- Kenntnis der User-ID des anderen Benutzerkontos
  - Diese werden in URLs verwendet und an verschiedenen Orten preisgegeben, z B. in Browserverläufen und Server-Logs, siehe auch RFC 3986

3.2.2 Empfehlung

- Echtheit von JWTs mit einer Signaturprüfung sicherstellen (über alle Werte, die im JWT hinterlegt sind)
- Signaturen vor dem Ausführen jeglicher Aktionen validieren und JWT mit ungültiger oder fehlender Signatur ablehnen

3.2.3 Technische Details

Ein gültiges JWT lässt sich so manipulieren, dass ein anderer Benutzer damit impersonisiert werden kann. Darüber kann das Benutzerkonto beliebiger anderer Benutzer – auch außerhalb der eigenen Organisation – übernommen werden. Für die Übernahme muss lediglich die User-ID des zu kompromittierenden Benutzerkontos bekannt sein. Diese wurde an mehreren anderen Stellen preisgegeben.

Das Verhalten ist auf eine fehlerhafte Signaturprüfung des JWT zurückzuführen.

Nachfolgend sind die Schritte beschrieben, mit denen der Benutzer **pentest01** (zugehörig zu MB-Organization**1**) Zugriff auf das Benutzerkonto von **pentest04** (Eigentümer von MB-Organization**2**) erhält und dieses damit übernehmen kann.

Abbildung 6 - Analyse eines JWT

Die beschriebenen Schritte können für alle Endpunkte genutzt werden, für die die Authentifizierung mittels JWT abgewickelt wird.

Beispielsweise ist darüber auch eine Passwortänderung des Benutzers möglich, siehe auch 3.5 FIN-05: Ausführung von sensiblen Aktionen ohne Identitätsüberprüfung.

1. **Vorbereitung – User-ID von pentest04 in Erfahrung bringen**: Die User-ID (`asd123-a12f-a12f-a12f-09asdoaijs`) von Benutzer *pentest04* muss bekannt sein. Da die User-IDs teils als Bestandteil von URLs verwendet werden (bspw. https://example.com/consumer/12322112/users/asd123-a12f-a12f-a12f-09asdoaijs/permissions), betrachten wir es als möglich, diese in Erfahrung zu bringen. Das liegt daran, dass URLs an verschiedenen Orten preisgegeben werden, z. B. in Browserverläufen und Server-Logs, siehe auch RFC 3986.

2. **Anmeldung mit eigenem Benutzerkonto pentest01**: Normal anmelden an der Anwendung mit dem Benutzerkonto *pentest01*. Daraufhin wird ein JWT ausgestellt. Das JWT wird bei API-Aufrufen im HTTP-Header `Authorization` zur Authentifizierung verwendet. Die Abbildung zur Analyse des JWT zeigt dessen Inhalt in der Base64-dekodierten Version. Die nebenstehende Abbildung zeigt exemplarisch den legitimen HTTP-Aufruf eines Endpunkts von Organisation1 mit einem gültigen JWT für Benutzer *pentest01*.

3. **Manipulation des JWT**: Mittleren Teil des JWT (begrenzt durch Punkte `.`, in vorheriger Abbildung markiert) kopieren, Base64-dekodieren, die ID im Parameter `sub` ersetzen durch die von Benutzer pentest04 (`asd123-dead-beef-a12f-09asdoaijs22`) und wieder Base64-kodieren. Im HTTP-Request den mittleren Teil des JWT durch die manipulierte Variante ersetzen.

4. **Impersonisieren von Benutzer pentest04**: Mit dem manipulierten JWT ist nun der Zugriff auf Ressourcen von MB-Organization2 möglich, für die der Benutzer *pentest04* (und *nicht* pentest01) berechtigt ist. Ein Beispiel ist der Aufruf der Kundendatenbank MB-Organization2. Der erfolgreiche Aufruf ist in der nebenstehenden Abbildung zu sehen. Die Kundennummer für den Aufruf des Endpunkts muss ebenfalls bekannt sein und kann vermutlich über eine Internet-Recherche leicht in Erfahrung gebracht werden.

Abbildung 7 - Ändern des Werts des JWT-Parameters `sub` in User-ID von pentest04

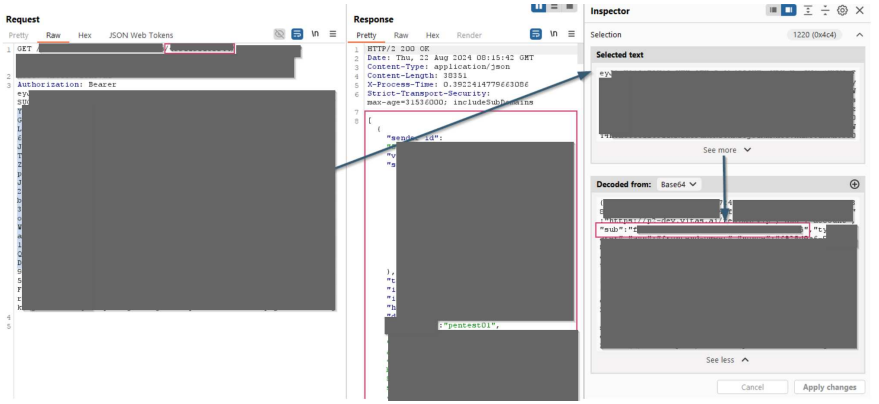

Abbildung 8 - Aufruf von Daten von MB-Organization2 mit gefälschtem JWT

3.3 FIN-03: Einschleusen bösartiger Inhalte in Website-Cache

Betroffen: CVSS v3.1: 7.5 (High)

- https://example.com/

3.3.1 Übersicht

Inhalte der Website werden anhand von Benutzereingaben erstellt. Anschließend werden die so generierten Inhalte im Cache gespeichert und an andere Website-Besucher ausgeliefert. Angreifer könnten das nutzen, um bösartige Inhalte einzuschleusen und dadurch andere Besucher anzugreifen.

Mögliche Folgen einer erfolgreichen Ausnutzung

- Platzieren von bösartigen Inhalten, die an andere Website-Besucher ausgeliefert werden, mit folgenden Auswirkungen:
  - Manipulieren der Darstellung einer Website
  - Ausführen von bösartigem JavaScript-Code
  - Weiterleiten von Benutzern auf bösartige Websites

Beispiele für Voraussetzungen für eine Ausnutzung

- Angreifer muss das Caching-Verhalten der Website genau untersuchen
- Bösartige Inhalte müssen bei Ablauf des Caches erneut platziert werden
- Benutzer müssen auf der Homepage auf einen vom Angreifer manipulierten Cache-Eintrag navigieren
  - Könnte potenziell die gesamte Website sein oder durch das Versenden von URLs zu manipulierten Seiten geschehen

3.3.2 Empfehlung

- Evaluieren, ob die Seiteninhalte ohne die Benutzereingaben aus den Headern *X-Forwarded-Host* und *X-Forwarded-Prefix* generiert werden können
- Falls das nicht möglich ist, eine der folgenden Maßnahmen ergreifen:
  - Eventuelle Benutzereingaben in den Headern von Systemen, die an der Kommunikation beteiligt sind, mit vordefinierten Werten überschreiben
  - Erhaltene Werte anhand einer Liste gültiger Werte filtern
- Nach weiteren möglichen Headern, Parametern oder anderen Stellen suchen, die ebenfalls zur Generierung der Seiteninhalte genutzt werden und von Benutzern beeinflusst werden können
- Mögliche temporäre Lösung, bis die Schwachstelle behoben ist:
  - Caching-Lösung deaktivieren (dadurch gehen jedoch auch die Vorteile des Caches verloren)

3.3.3 Technische Details

Der Inhalt des Server-Caches wird basierend auf den Headern *X-Forwarded-Host* und *X-Forwarded-Prefix* generiert. Beide Header können von Benutzern in HTTP-Anfragen gesetzt und mit bösartigem Inhalt befüllt werden. Der so im Cache platzierte Inhalt wird dann beim Aufruf der Website durch andere Besucher an diese ausgeliefert. Dieser Angriff ist auch unter Web Cache Poisoning bekannt.

Der Cache hat nach unserer Beobachtung eine Gültigkeit von 200 Sekunden. Nach dieser Zeit wird der Cache mit dem nächsten Aufruf der Website erneuert. Ein Angreifer muss bösartige Inhalte deshalb nach Ablauf der Zeit erneut platzieren, bevor ein legitimer Benutzer die Website aufruft.

Um den regulären Betrieb der Website nicht zu stören, erstellten wir durch den URL-Parameter *cachebuster* einen eigenen Cache-Eintrag. Die Cache-Einträge der Website, welche für den regulären Aufruf benötigt wurden, blieben so unberührt. Angriffe sind aber auch auf die regulären Cache-Einträge möglich, wodurch jeder legitime Website-Besucher betroffen wäre.

Die nachfolgenden Abschnitte beschreiben mögliche Angriffe, die sich aus dem Cache-Verhalten ergeben.

Ausführen von bösartigem JavaScript-Code

Beim Aufruf der JavaScript-Ressource global.js verwendet die Anwendung den HTTP-Header *X-Forwarded-Host*, um Teile des Codes zu generieren. So lässt sich die Funktionalität des Skripts erweitern und bösartiger Code auf jede Seite bringen, die das Skript verwendet.

Das Ausführen des eingeschleusten Codes kann beim Aufruf der URL https://example.com provoziert werden. Das manipulierte Skript ist auf zahlreichen Seiten eingebunden, wodurch eine Ausführung wahrscheinlicher wird.

Um den produktiven Betrieb nicht zu stören, haben wir auch hier den Parameter `cachebuster` genutzt. Dieser wird vom verwendeten Proxy-Tool *Burp Suite* beim Laden der JavaScript-Ressource automatisch hinzugefügt. Ein Angreifer müsste den richtigen Cache-Eintrag des Skripts manipulieren, um den eingeschleusten Code erfolgreich auszuführen.

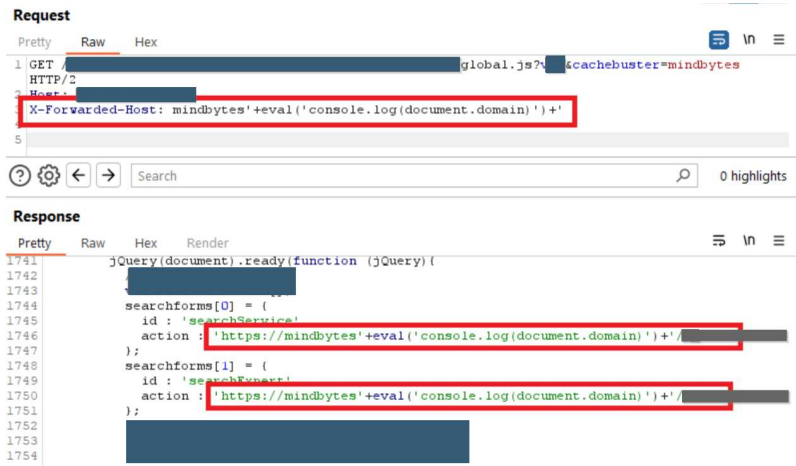

Abbildung 9 - Einschleusen von JavaScript-Code in den Cache

Weiterleiten auf bösartige Websites

Beim Aufruf der Startseite und anderer existierender Verzeichnisse, wie beispielsweise https://www.example.com, verwendet die Anwendung den Header *X-Forwarded-Host*, um die korrekte Weiterleitung des Benutzers zu generieren. Die Weiterleitung wird dabei auch in den Website-Cache gelegt. Mit einem manipulierten HTTP-Header lässt sich die Weiterleitung auf die angegebene Domäne und damit zu einer potenziell bösartigen Website erreichen.

Eine Manipulation, wie sie in der Abbildung vorgenommen wird, führt dazu, dass andere Benutzer, die die URL https://www.example.com/?cachebuster=mindbytes aufrufen, auf die Website der MindBytes weitergeleitet werden.

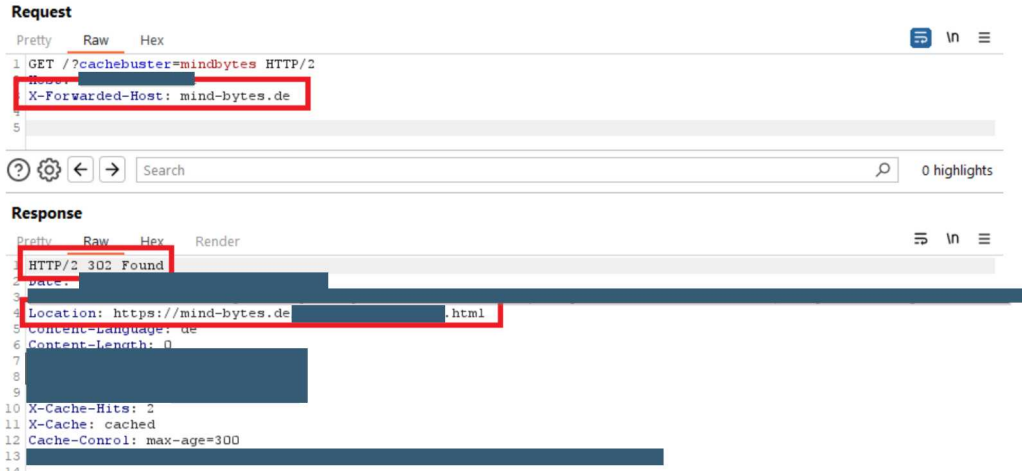

Abbildung 10 - Einschleusen einer bösartigen Weiterleitung in den Cache

Manipulieren des Webinhalts

Für die Erstellung des Seiteninhalts wurden sowohl der Header *X-Forwarded-Host* als auch *X-Forwarded-Prefix* genutzt. Konkret wurden die angegebenen Inhalte auf der Website unter anderem als HTML-Tag `<base href="https://<forwarded-host>/<forwarded-prefix>` eingebettet. Dieses Meta-Tag weist den Browser an, alle relativen URLs von der dort angegebenen Domäne beziehungsweise URL zu laden.

Beim Aufruf der manipulierten Website wurde die so platzierte Domäne jedoch vom Browser blockiert. Das lag an der konfigurierten `Content-Security-Policy`, die das Laden von Ressourcen nur von der eigenen und ausgewählten Domänen erlaubt, nicht jedoch von beliebigen anderen Domänen.

Abbildung 11 - Einschleusen von manipuliertem Inhalt

3.4 FIN-04: Spring Boot Actuator aktiv

Betroffen: CVSS v3.1: 7.5 (High)

- https://example.com/

3.4.1 Übersicht

Der Actuator stellt Funktionen für die Überwachung der Anwendung bereit und ist Teil des Java Spring Frameworks. Über den Spring Boot Actuator lassen sich detaillierte Information auslesen, die für Angreifer wertvoll sind. Im konkreten Fall können die Sitzungen anderer Benutzer übernommen werden.

Mögliche Folgen einer erfolgreichen Ausnutzung 🔥🔥🔥🔥🔥

- Auslesen sensibler Informationen
- Zugriff auf Session-Cookies anderer Benutzer und anschließender Zugriff auf die Anwendung mit deren Berechtigungen
- Weitere Angriffe, die durch das Nutzen der sensiblen Informationen möglich werden, wie beispielsweise das Verwenden von offenbarten Zugangsdaten

Beispiele für Voraussetzungen für eine Ausnutzung 🎲🎲🎲🎲🎲

- Für die meisten Endpunkte ist ein gültiges Benutzerkonto nötig
- URL muss bekannt sein – kann leicht erraten werden
- Angreifer muss ggf. in der richtigen Netzwerkposition sein, um die Informationen nutzen zu können
  - Beispielsweise muss die Datenbank erreichbar sein, um Zugangsdaten nutzen zu können

3.4.2 Empfehlung

- Evaluieren, ob der Actuator benötigt wird
- Actuator deaktivieren oder Funktionalität einschränken
- Mögliche offenbarte sensible Informationen wie Datenbank-Zugangsdaten ändern

3.4.3 Technische Details

Unter der URL https://example.com/actuator ist der Spring Boot Actuator erreichbar und sind alle verfügbaren Funktionen auflistbar.

Insbesondere ist der Endpunkt /actuator/heapdump für Angreifer interessant. Hierüber lässt sich ein Abbild des Java Heaps erstellen. Der Heap ist der Bereich des Arbeitsspeichers, in der die Anwendung ihre Objekte speichert. Beispielsweise sind hier Umgebungsvariablen aber auch Objekte zu aktuellen HTTP-Anfragen gespeichert.

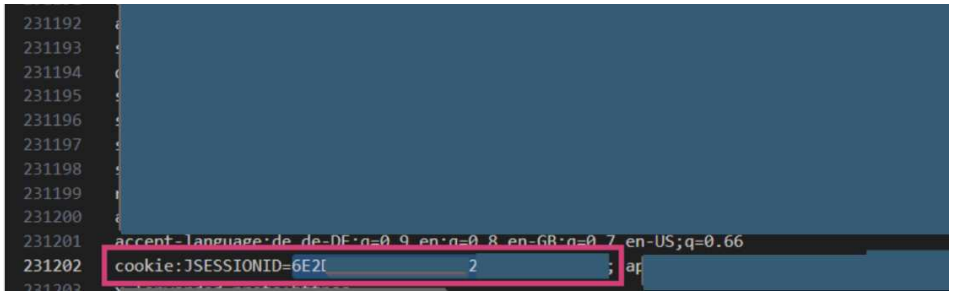

Abbildung 12 - Einsicht eines Session-Cookies innerhalb des exportierten Heaps

Die Abbildung zeigt den mit dem regulären Benutzer `pentest1` exportierten Heap in einem Texteditor. Der Ausschnitt zeigt, dass im Heap die Details zu aktuellen HTTP-Anfragen enthalten sind. Unter anderem konnten wir so auf die Session-Cookies `JSESSIONID` von anderen aktiven Benutzern zugreifen.

Das ausgelesene Session-Cookie ist gültig und kann im Browser genutzt werden. Somit ist der Zugriff auf die Anwendung mit einem anderen Benutzer möglich.

Im exportierten Heap waren auch weitere sensible Informationen enthalten, beispielsweise die Werte der Umgebungsvariable `db.url`, `db.username` und `db.password`. Damit wäre vermutlich der Zugriff auf die Datenbank möglich, falls diese netzwerktechnisch erreichbar ist.

3.5 FIN-05: Ausführung von sensiblen Aktionen ohne Identitätsüberprüfung

Betroffen: CVSS v3.1: 5.6 (Medium)

- https://example.com/

3.5.1 Übersicht

Sensible Aktionen können durchgeführt werden, ohne dass zuvor die Legitimität des Nutzers bestätigt wird. Konkret können Benutzer ihr Passwort ohne erneute Passworteingabe ändern. Das erhöht das Risiko der Übernahme von Benutzerkonten im Falle unbeaufsichtigter Sitzungen.

Mögliche Folgen einer erfolgreichen Ausnutzung

Angreifer mit Zugriff auf eine aktive Benutzersitzung können ungehindert sensible Aktionen durchführen:

- Im konkreten Fall kann das Passwort geändert und so die Kontrolle über das Konto erlangt werden
- Der rechtmäßige Benutzer würde (vorübergehend) Zugriff auf das Konto verlieren
- Angreifer könnte jede Aktion durchführen, die die Berechtigungen des Kontos zulassen
- Ermöglicht häufig den Zugriff auf sensible Informationen oder das Löschen des Kontos

Beispiele für Voraussetzungen für eine Ausnutzung

Mehrere Angriffsmöglichkeiten, bei denen der fehlende Schutz ausgenutzt werden kann:

- Angreifer nutzt z. B. eine Cross-Site-Scripting-Schwachstelle, um die sensible Aktion aus der Ferne durchzuführen
- Benutzer lässt eine authentifizierte Sitzung unbeaufsichtigt, z. B. bei gemeinsam genutzten Computern

3.5.2 Empfehlung

- Vor der Ausführung sensibler Aktionen das Passwort des Benutzers oder ein anderes Authentifizierungsmittel prüfen

3.5.3 Technische Details

Die Abbildung zeigt, dass das Passwort des Benutzers geändert werden kann, ohne das alte Passwort angeben zu müssen.

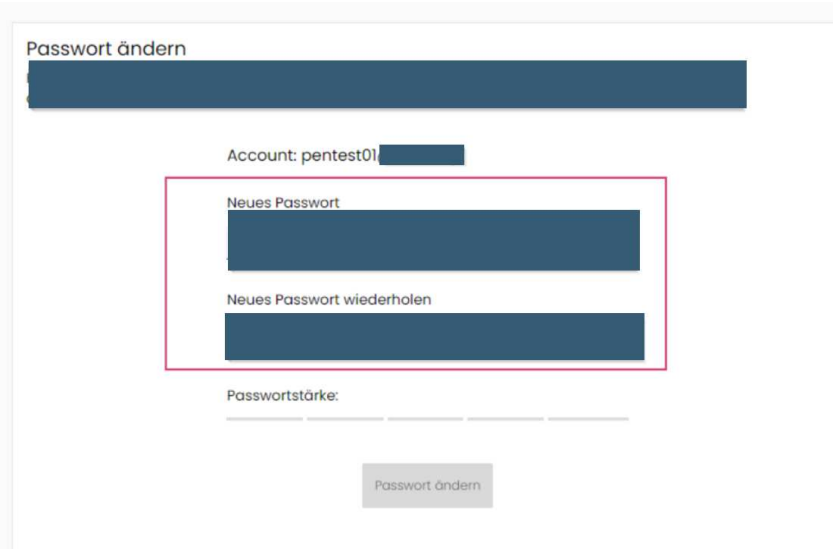

Abbildung 13 - Passwortänderung ohne Kenntnis des alten Passworts

3.6 FIN-06: Fehlende HTTP-Security-Header

Betroffen: CVSS v3.1: 4.2 (Medium)

- https://example.com/

3.6.1 Übersicht

Fehlende HTTP-Security-Header stellen an sich keine Schwachstelle dar. Das Konfigurieren solcher Header kann aber die Sicherheit von Webanwendungen erhöhen. Typische Angriffe werden so erschwert oder ganz verhindert. Die Header sind als zusätzliche Sicherheitsmaßnahme zu sehen und ersetzen nicht die sichere Programmierung von Webanwendungen.

Mögliche Folgen einer erfolgreichen Ausnutzung

- Ausnutzung typischer Schwachstellen, beispielsweise Clickjacking oder Cross-Site-Scripting, ist ohne Security-Header einfacher möglich

Beispiele für Voraussetzungen für eine Ausnutzung

- Fehlende Header sind sicherheitsrelevant, wenn Schwachstellen in Anwendungen vorhanden sind und/oder Interaktion von Benutzern stattfindet

3.6.2 Empfehlung

- Falls möglich, die fehlenden Security-Header auf allen aufgeführten Websites (siehe „Technische Details“) setzen
- Empfehlungen zur Konfiguration finden sich im jeweiligen Abschnitt der OWASP Cheat Sheets:
  Strict-Transport-Security | Content-Security-Policy | X-Frame-Options, | X-Content-Type-Options | Referrer-Policy | Permissions-Policy
- Header-Konfiguration bei folgenden Websites verbessern:
  - https://example.com
    - Bei der Content-Security-Policy auf die Direktiven *unsafe-eval* und *unsafe-inline* verzichten

- Eine Auswertung der konfigurierten Content-Security-Policy kann mit dem CSP Evaluator vorgenommen werden

3.6.3 Technische Details

Die folgende Tabelle gibt einen Überblick über die konfigurierten Security-Header auf den Websites.

- Mit ✓ markierte Header sind gesetzt und werden als angemessen eingestuft.
- Mit 👓 markierte Header sind gesetzt, die Konfiguration sollte jedoch überprüft und verbessert werden.
- Nicht markierte Header sind nicht gesetzt.

Website	Strict-Transport-Security	Content-Security-Policy	X-Frame-Options	X-Content-Type-Options	Referrer-Policy	Permissions-Policy
https://example.com	✓	👓				

4 Projektrahmen

4.1 Involvierte Personen

Name	Rolle	Mail-Adresse
Simon Holl	Projektleitung & Durchführung	hallo@mind-bytes.de
Christian Stehle	Durchführung	hallo@mind-bytes.de
Nina Wagner	Durchführung	hallo@mind-bytes.de
Anja Neudert	Review	
Max Musterfrau	IT-Leitung	max.musterfrau@musterfirma.de

4.2 Testzeitraum

02.01.25 - 10.01.25

4.3 Testgegenstand

Asset-Typ	Wert	Beschreibung
Webanwendung	https://example.com/	Kundenportal

4.4 Zugriffsweg

Der Zugriff erfolgte über das Internet.

4.5 Bereitgestellte Benutzerkonten

Benutzerkonto	Rolle/Rechte
pentest01	Rolle Benutzer
pentest02	Rolle Benutzer
pentest03	Rolle Administrator
pentest04	Rolle Administrator

4.6 Bereitgestellte Informationen

Um zielgerichtete und effiziente Prüfungen zu ermöglichen, wurden folgende Daten bereitgestellt:

- Quellcode der Webanwendung

5 Anhang

5.1 Erläuterungen Bewertungsskalen

	Common Vulnerability Scoring System (CVSS)	MindBytes-Score
Erläuterung	▪ Standardisiertes Bewertungssystem für die Schwere von Sicherheitslücken in Software und Systemen ▪ Technische Bewertung ▪ De facto Industrie-Standard	▪ Bewertungssystem der MindBytes mit risikobasiertem Ansatz und Fokus auf (potenziellem) Schaden und Wahrscheinlichkeit ▪ Wahrscheinlichkeit bedeutet in diesem Kontext, wie einfach eine Schwachstelle ausnutzbar ist ▪ Der Score basiert auf der CVSS-Bewertung und lässt darüber hinaus die Anzahl und Wichtigkeit der betroffenen Systeme einfließen
Bewertungsskalen	Skala von 0 (Info) bis 10 (kritisch) zur Einstufung der Schwere einer Schwachstelle	Skala von 0-5 zur Bewertung von Schaden und Wahrscheinlichkeit

6 Änderungsverzeichnis

Version	Datum	Änderung	Wer
1.0	20.01.25	Freigabe	Christian Stehle

7 Disclaimer

Dieses Projekt wurde durchgeführt, um die Sicherheit der im Fokus liegenden Komponenten zu bewerten und Schwachstellen aufzudecken.

1. Bei diesem Test handelt es sich um eine Momentaufnahme und keine fortlaufende Sicherheitsüberwachung. Die Sicherheitslage kann sich im Laufe der Zeit ändern, beispielsweise durch Veränderungen an den Komponenten, preisgegebenen Informationen, neue Angriffstechniken oder Schwachstellen.
2. Das Projekt wurde innerhalb eines begrenzten Zeitrahmens durchgeführt. Dies kann dazu führen, dass nicht alle potenziellen Schwachstellen und preisgegebenen Informationen identifiziert wurden.
3. Auch wenn das Projekt mit großer Sorgfalt durchgeführt wurde, sind False-Positives nicht auszuschließen.

8 Impressum

MindBytes GmbH | Probststraße 15 | 70567 Stuttgart

+49 711 20709567 | hallo@mind-bytes.de | https://mind-bytes.de

Amtsgericht Stuttgart, HRB 790784 | USt-IdNr: DE363069855

vertreten durch die **Geschäftsführung Christian Stehle, Nina Wagner, Simon Holl**