APOLLON
SECURITY

Mandant AG

Penetration Test einer Webanwendung
Beispiel-Report

Autor: Apollon Security GmbH
Version: 1.0
Datum: 27.10.2023

Rheinpromenade 9, 40789 Monheim am Rhein
www.apollon-security.com
02173 – 265 3550
kontakt@apollon-security.com

Versionsverlauf

Ver.	Datum	Änderungen	Autor
1.0	27.10.2023	Erstellung	Max Mustermann

Kunde:

Mandant AG

Erstellt von:

Apollon Security GmbH
Rheinpromenade 9
40789 Monheim am Rhein

Inhaltsverzeichnis

1. Übersicht

Dieses Dokument beinhaltet eine detaillierte Aufstellung der Ergebnisse des von der Apollon Security GmbH durchgeführten Penetrationstests einer Webanwendung für die Mandant AG.

Projektbeteiligte:

Name	Rolle
Max Mustermann	Senior Penetration Tester und Projektleiter
Maria Müller	Penetration Tester

1.1 Ziel, Umfang und Methodik

Das Ziel des Penetrationstests bestand darin, Schwachstellen in der Webanwendung zu identifizieren und zu bewerten, die beispielsweise zu einem unberechtigten Zugriff oder zu Datendiebstahl führen könnten.

Die Bewertung des Risikos einer Schwachstelle setzt sich dabei aus ihrem Schaden und der Komplexität zusammen. Eine Erläuterung der Risikobewertung mit Angaben zu den einzelnen Risikostufen wird im Anhang B beschrieben.

Der Penetrationstest fand per Fernzugriff vom 23.10.2023 bis zum 26.10.2023 statt. Die angewandte Methodik zur Ermittlung von Schwachstellen basierte auf dem OWASP Web Security Testing Guide in der aktuellen Version 4.2 (https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing).

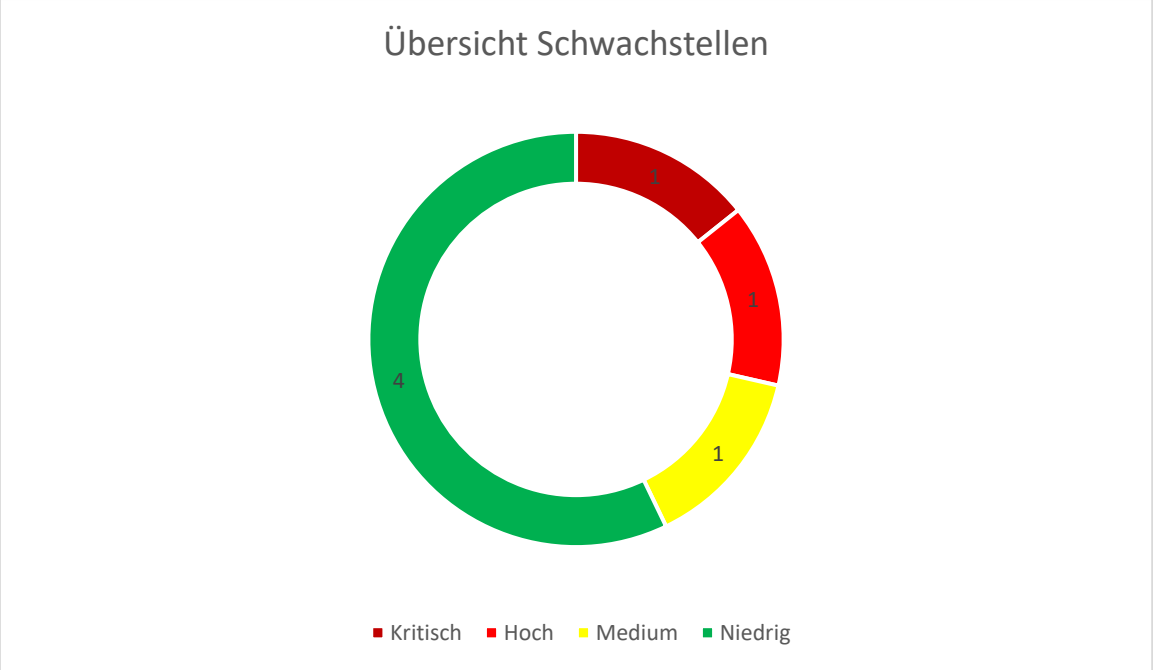
Auf der Grundlage der in diesem Bericht ermittelten Ergebnisse kann die Mandant AG angemessene Maßnahmen ergreifen, um die identifizierten Schwachstellen zu beheben und die allgemeine Sicherheitslage damit zu verbessern.

2. Management Zusammenfassung

Die Mandant AG beauftragte Apollon Security mit einem Penetrationstest ihrer Webanwendung. Das Ziel dieser Überprüfung war es, vorhandene Sicherheitslücken in der bereitgestellten Testumgebung festzustellen.

Für den Penetrationstest wurden zusätzlich zwei Testbenutzer erstellt (Standard- und Admin-Rechte), um den authentifizierten Bereich der Webanwendung prüfen zu können.

Insgesamt wurden **7** Schwachstellen identifiziert. Eine Aufteilung der einzelnen Lücken nach Risiko wird im folgenden Diagramm aufgezeigt:



Im Folgenden werden die Sicherheitslücken mit einem kritischen und hohen Risiko zusammengefasst:

- Durch eine fehlende Prüfung von Eingabedaten kann ein Angreifer direkt auf die zugrundeliegende Datenbank der Webanwendung zugreifen und beispielsweise Kundendaten einsehen oder manipulieren.
- Ein Angreifer mit Standardrechten kann durch eine weitere unzureichende Prüfung von Eingabedaten Schadcode in die Webanwendung einfügen, um beispielsweise auf sensible Informationen anderer Benutzer zuzugreifen oder fremde Inhalte in die Anwendung einzuschleusen.

Apollon empfiehlt die in diesem Bericht adressierten Schwachstellen zeitnah zu beheben. Dabei sollte nach Möglichkeit ein risikobasierter Ansatz verfolgt

werden. Im Anschluss daran wird ein erneuter Penetration Test empfohlen, um die behobenen Schwachstellen auf ihre Wirksamkeit zu prüfen.

Eine detaillierte Beschreibung der oben genannten Schwachstellen, die dazugehörigen Risiken und Empfehlungen zur Behebung dieser, befinden sich in den weiteren Kapiteln dieses Berichts.

3. Testumfang

Der Penetrationstest der Webanwendung erfolgte auf der zur Verfügung gestellten Testumgebung, die aus dem Internet über folgende URL erreichbar war:

https://mandanten-anwendung-test.de

Zur Überprüfung des authentifizierten Bereichs wurden folgende Benutzerkonten bereitgestellt:

Name	Rolle
Testuser	Benutzer mit Standardrechten
Testadmin	Benutzer mit administrativen Rechten

3.1 Einschränkungen

Folgende Einschränkungen in Bezug auf den durchgeführten Penetrationstest sollten beachtet werden:

i) Gezielte Angriffe auf die Verfügbarkeit der Anwendung (Denial of Service) waren kein Teil dieser Überprüfung

4. Methodik

Die vorliegende Methodik wird zur Durchführung des Penetrationstests in vier Phasen angewendet:

- Phase 1: Vorbereitung

Der Test wird vorbereitet, indem die Definition der Zielsetzung, die Festlegung des Zeitrahmens und die Risiken bei der Prüfung gegengeprüft werden.

Außerdem werden die Rahmenbedingungen für die Durchführung des Tests festgelegt.

- Phase 2: Informationsbeschaffung

In der Phase der Informationsbeschaffung werden einzelne Informationen über die Zielsysteme und die Netzwerkinfrastruktur beschafft. Dies kann mithilfe von automatischen Schwachstellenscans erfolgen, aber auch manuellen Prüfungen von technischen Beschaffenheiten passieren. Dabei wird unter anderem auch die Verschlüsselung der Webseite geprüft oder die erreichbaren Ports.

- Phase 3: Bewertung der Informationen

In der Bewertung werden die vertraglich definierten Ziele, die potenziellen Auswirkungen eines Angriffes auf das System und der Aufwand für die Bewertung der Schwachstellen durch Ausnutzung dieser betrachtet. Basierend auf dieser Evaluation werden dann die Angriffsziele für Phase 4 bestimmt.

- Phase 4: Ausnutzung von Schwachstellen

Mithilfe der Informationen aus Phase 2 werden nun konkrete Eindringungsversuche zur Verifizierung der Möglichkeit einer aktiven Ausnutzung von Schwachstellen des Systems durchgeführt. Es wird mithilfe technischer Werkzeuge und Methoden probiert, Sicherheitsvorrichtungen zu umgehen, höhere Berechtigungen zu erlangen, um schlussendlich an vertrauliche Informationen zu kommen.

- Phase 5: Analyse und Dokumentationen des Tests

Die in den vorherigen Phasen erlangten Informationen, gefundenen Schwachstellen und Sicherheitsprobleme werden abschließend gesammelt und in einem Bericht aufgeführt. Zusätzlich wird beschrieben, welche Voraussetzungen eine aktive Ausnutzung dieses Sicherheitsproblems hat. Auf Basis dieser Ergebnisse werden dann konkrete Maßnahmen und Empfehlungen ausgearbeitet.

5. Eingesetzte Werkzeuge

Für diesen Penetrationstest wurden die folgenden Werkzeuge verwendet.

Name	Beschreibung
Tenable Nessus	Netzwerkbasierter Schwachstellen-Scanner
Burp Suite Professional Edition	Software zum Testen der Anwendungssicherheit
Nmap	Netzwerkbasierter Portscanner
SoapUI	Testen von Webservices
OpenVAS	Netzwerkbasierter Schwachstellen-Scanner
Skipfish	Web-Application-Scanner
Sqlmap	Werkzeug zum Ausnutzen von SQL-Injection-Schwachstellen

Die genutzten Werkzeuge wurden dabei je nach Kenntnisstand des durchführenden Testers ausgewählt und für deren vorgesehenen Verwendungszwecke verwendet. Alle durch die Werkzeuge hinterlassen Restdateien wurden nach Abschluss des Tests von den Systemen entfernt.

6. Übersicht der Schwachstellen

Die folgende Tabelle zeigt eine Übersicht der ermittelten Sicherheitslücken.

ID	Schwachstelle	Risiko	Status
Webapp-01	SQL-Injection	Kritisch	Offen
Webapp-02	Cross-Site Scripting – Persistent	Hoch	Offen

Webapp-03	Lokales Caching von Anwendungsdaten	Mittel	Offen
Webapp-04	Cookie ohne "Secure" Eigenschaft	Gering	Offen
Webapp-05	Unzureichende Fehlerbehandlung	Gering	Offen
Webapp-06	Versionsinformationen in HTTP-Headern	Gering	Offen
Webapp-07	Veraltete Software in Betrieb	Gering	Offen

7. Details

Webapp-01: SQL-Injection

Schaden	Komplexität	Risiko	CVSS:3.1	Status
Kritisch	Einfach	Kritisch	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:N	Offen

Angriffsvoraussetzungen
Der Angreifer muss die Webanwendung über das Internet erreichen können.

Beschreibung
Die Webanwendung ist innerhalb des Parameters “language“ für SQL-Injection anfällig. Dieser Parameter legt die Sprache der Anwendung fest und wird auf der Startseite der Anwendung als Teil der URL übertragen.

SQL-Injection erlaubt es einem Angreifer in der Webanwendung eigene Datenbankbefehle durch eine fehlende Eingabevalidierung auszuführen und somit beispielsweise Daten zu extrahieren oder zu modifizieren.

Beispielsweise kann ein Angreifer durch folgende Modifikation des Parameters (in gelb markiert) unterschiedliche Antworten der zugrundeliegenden Datenbank provozieren und über diesen Weg (True- bzw. False-Statements) Daten extrahieren:

GET /secpages/login.aspx?language=de%27%20and%20%27a%27=%27a HTTP/1.1

Host: mandanten-anwendung-test.de

Zur Verdeutlichung:
Der in gelb eingefügte Payload sieht in seiner ursprünglichen Form (ohne URL-Kodierung) wie folgt aus: ' and 'a'='a

Durch eine fehlende Eingabeprüfung wird dieser exemplarische Payload ungefiltert an die Datenbank weitergereicht und erzeugt ein so genanntes True-Statement, wohingegen z. B. der Payload ' and 'a'='b ein False-Statement verursacht.

Durch diese Angriffstechnik werden unterschiedliche Antworten der Webanwendung zurückgegeben (je nach True- oder False-Statement), die wiederum ausgenutzt werden können, um Daten zu extrahieren. Beispielsweise kann durch folgende Eingabe abgefragt werden, ob der erste Buchstabe des aktuellen Datenbankbenutzers ein “m“ ist:

‘ and substring((select user),1,1) = ‘m

Die oben gezeigte Abfrage resultiert in einem True-Statement. Ein Angreifer kann dieses Beispiel weiterhin für sich ausnutzen, um sukzessive über True- und False-Statements den vollständigen Namen des Datenbankbenutzers auszulesen.

Apollon konnte über diese Schwachstelle unter anderem sämtliche in der Datenbank gespeicherten Kundendaten auslesen. Aus Vertraulichkeitsgründen werden diese Details nicht im Bericht abgebildet, können aber bei Bedarf gesondert zur Verfügung gestellt werden.

Empfehlung
Zur Behebung der Schwachstelle sollten so genannte “Prepared Statements“ in der Anwendung implementiert werden. Diese definieren Platzhalter in der jeweiligen Datenbankabfrage der Webanwendung, wodurch sichergestellt wird, dass Sonderzeichen wie ein Hochkomma als Zeichenwert und nicht als Steuerzeichen interpretiert werden. Durch diese Maßnahme werden mögliche Änderungen in der zugrundeliegenden SQL-Abfrage verhindert.

Falls SQL-Abfragen mittels “Prepared Statements“ nicht umsetzbar sind (z. B. in ORDER BY Strukturen) sollte eine geeignete Whitelist implementiert werden, die beispielsweise nur Groß- und Kleinbuchstaben als Eingabe erlaubt.

Weitere Details zur Mitigation von SQL-Injection-Angriffen können auf folgende URL eingesehen werden:
https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

Webapp-02: Cross-Site Scripting – Persistent

Schaden	Komplexität	Risiko	CVSS:3.1	Status
Hoch	Moderat	Hoch	CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:U/C:L/I:L/A:N	Offen

Angriffsvoraussetzungen
Der Angreifer benötigt einen authentifizierten Zugriff mit Standardrechten auf die Webanwendung.

Beschreibung
Die beiden Eingabefelder “Firma“ und “Anrede“ innerhalb der Stammdaten-Webseite sind für Cross-Site Scripting (XSS) anfällig.

XSS ist eine Angriffstechnik, bei der HTML oder Skriptcode wie JavaScript in die Anwendung eingefügt wird, und dann im Browserkontext eines Benutzers zur Ausführung kommt. Dadurch ist ein Benutzer der Anwendung einer Reihe von möglichen Angriffen ausgesetzt. Beispielsweise könnten Angreifer auf diese Art auf sensible Informationen des Benutzers zugreifen oder fremde Inhalte in die Anwendung einbinden.

Die entdeckten XSS-Schwachstellen sind persistent. Dies bedeutet, dass der eingefügte Schadcode von der Anwendung gespeichert und bei einem Besuch des betroffenen Bereichs ausgeführt wird.

Die Webanwendung verwendet die ASP.NET Request Validation, um das Einfügen von Skriptcode zu verhindern. Dieses Schutzkonzept kann jedoch über eine spezielle Codierung, die öffentlich bekannt ist, umgangen werden.

Die Apollon hat exemplarisch innerhalb der Stammdaten-Webseite im authentifizierten Bereich der Anwendung Skriptcode im Eingabefeld „Firma“ eingefügt. Im Folgenden wird die zugehörige POST-Anfrage inklusive der Codierung aufgezeigt:

Request

Pretty Raw Hex \n

```
1 POST /secpages/StammdatenEdit.aspx HTTP/1.1
2 Host: [redacted]
3 Cookie: ASP.NET_SessionId=lztb0zchplh5vlnkqkipatxt; .ASPXAUTH=
  59DC20B43D7B019CA7D56820179BCF6A422EFAB0410C51C68474432A4A47A8926
  6671D8077E6EB8DCDAE6511456EE2488AFB3FD780286AE4E1C55ED60B95B7F7B6
  C9C14A0950C8DD2C0B0FAF5393CB43C66B477AA4DA95BB94F95ADEE8C36D40E42
  385DC795F1F6694D176A2651257B6625820407BF2EA48CA7A57CE035BB0F95C44
  7A0D75B326062DC51EBBB3BB8CDB
4
5 ctl00_menu_RadMenu1_ClientState=&
  ctl00_menu_RadMenuLogout_ClientState=&
  ctl00%24ContentPlaceHolder%24txbFirma=
  Apollon+Security+%EF%BC%9Cimg%20src%3Dxxx%20onerror%3Dalert('Apol
  lon-Script-Code')%EF%BC%9E&
  ctl00%24ContentPlaceHolder%24txbFirmaZusatz=&
  ctl00%24ContentPlaceHolder%24ddlAnrede=Herr&
  ctl00%24ContentPlaceHolder%24txbStrasse=Rheinpromenade+9&
  ctl00%24ContentPlaceHolder%24txbPLZ=40789&
  ctl00%24ContentPlaceHolder%24txbOrt=Monheim+am+Rhein&
  ctl00%24ContentPlaceHolder%24txbTelefon=0217332653559&
  ctl00%24ContentPlaceHolder%24txbMobil=124&
  ctl00%24ContentPlaceHolder%24txbFax=&
  ctl00%24ContentPlaceHolder%24txbEMail=&
  ctl00%24ContentPlaceHolder%24txbEMailConfirm=&
  ctl00%24ContentPlaceHolder%24btnSave=Speichern
```

Der eingefügte Skriptcode ist dabei der folgende Auszug:

%EF%BC%9Cimg%20src%3Dxxx%20onerror%3Dalert('Apollon-Script-Code')%EF%BC%9E

Dieser beispielhafte Proof-of-Concept Code erzeugt ein Popup-Fenster im Browser-Fenster des Benutzers mit dem Inhalt "Apollon-Script-Code“.

Empfehlung

Generell sollte der Schutz vor XSS-Angriffen nicht nur durch das ASP.NET Request Validation–Konzept abgedeckt sein, da dieser durch öffentlich bekannte Methoden ausgehebelt werden kann.

Stattdessen sollte die Anwendung sicherstellen, dass jede Nutzereingabe einen Validierungsprozess durchläuft. Beispielsweise sollten Sonderzeichen von der Anwendung als HTML-Entities codiert werden, bevor sie von der Anwendung gespeichert bzw. wiedergegeben werden. Dadurch können die meisten bösartigen Eingaben entschärft werden.

Weitere Informationen zu ASP.NET Request Validation sowie den Methoden in ASP.Net, um Benutzereingaben zu validieren, können unter folgender URL eingesehen werden: https://owasp.org/www-community/ASP-NET_Request_Validation

Eine ausführliche Anleitung zur Behebung von XSS-Schwachstellen kann auf folgender URL eingesehen werden:
https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

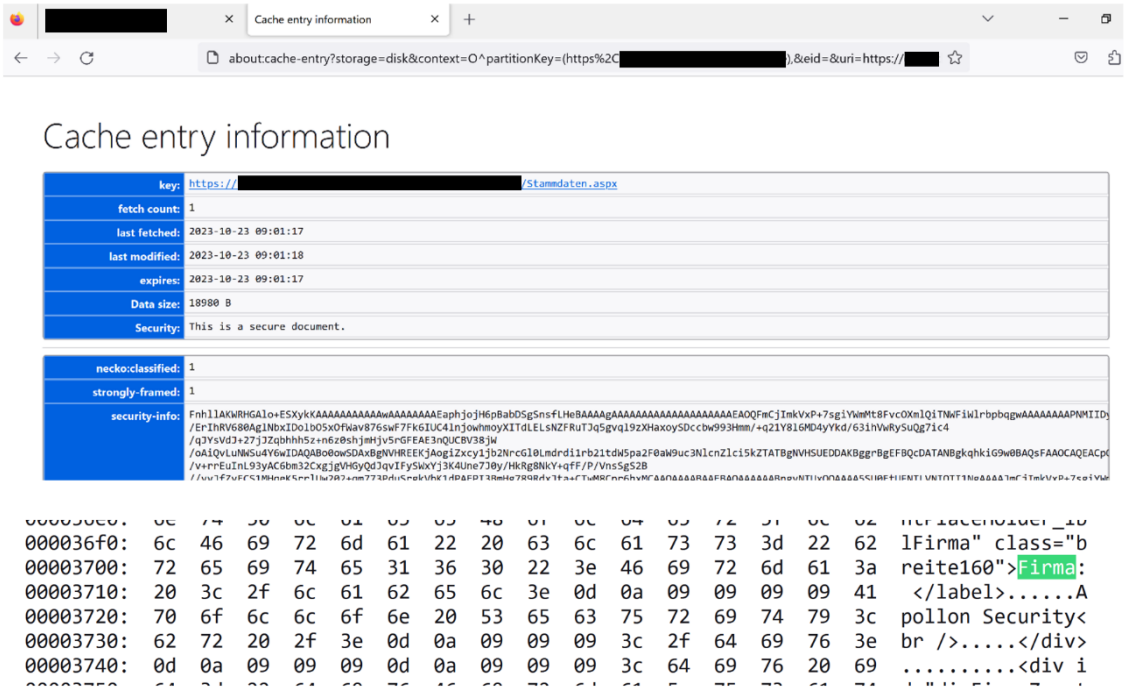
Webapp-03: Lokales Caching von Anwendungsdaten

Schaden	Komplexität	Risiko	CVSS:3.1	Status
Mittel	Moderat	Mittel	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	Offen

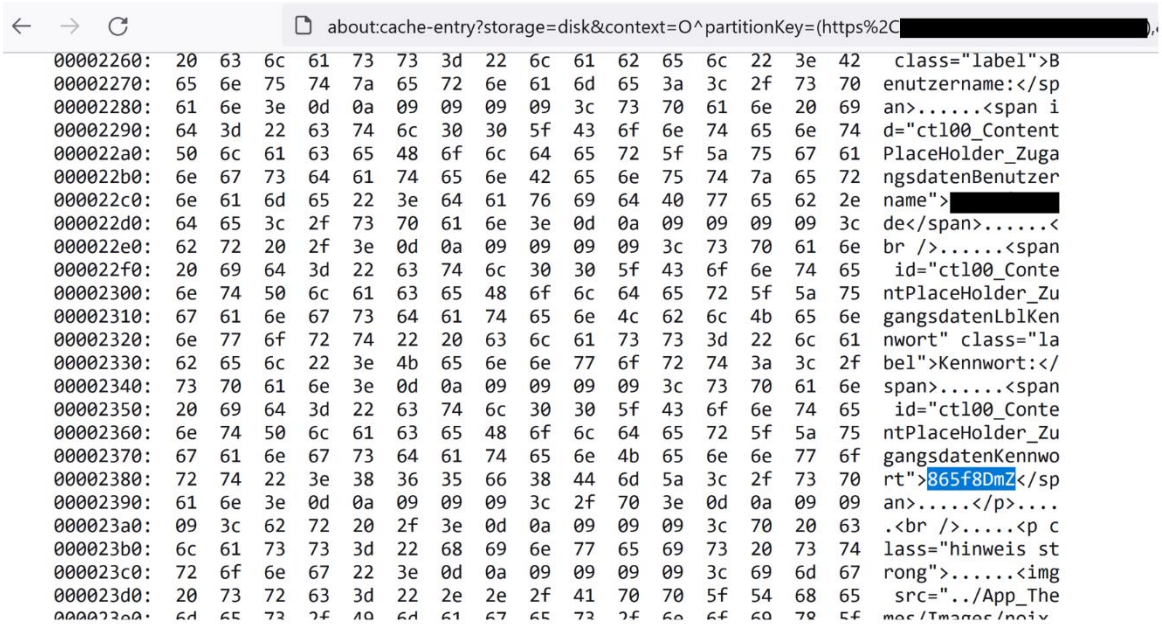
Angriffsvoraussetzungen
Der Angreifer benötigt Zugriff auf das System eines legitimen Benutzers der Webanwendung.

Beschreibung
Die Webseiten des authentifizierten Bereichs der Anwendung werden im Browser des Benutzers zwischengespeichert. Nachdem sich ein Benutzer von der Anwendung abgemeldet hat, können diese Seiten weiterhin über den lokalen Cache des Browsers geöffnet werden. In einem gemeinsam genutzten System sind diese Seiten unter Umständen anderen Benutzern zugänglich. Weiterhin können ebenfalls Angreifer, die sich unautorisierten Zugang auf ein System verschafft haben, diesen Cache auslesen und somit auf sensible Inhalte der Anwendung zugreifen.

Beispielsweise wird im folgenden Screenshot die im Cache des Firefox-Browsers gespeicherte Stammdaten.aspx Webseite mitsamt ihren Inhalten gezeigt:



Weiterhin kann ein Benutzer die Selbstregistrierung der Webanwendung benutzen, um sich einen eigenen Account anzulegen. Dabei wird ein automatisch erzeugtes Passwort generiert und nach Abschluss der Registrierung ausgegeben. Wie bereits im vorherigen Beispiel gezeigt, wird auch hier die Webseite (Benutzererfassung.aspx) im Browser-Cache mitsamt dem Benutzernamen und Passwort im Klartext abgespeichert:



Ein Angreifer kann zudem die XSS-Lücke (siehe Schwachstelle Webapp-02)

dafür ausnutzen, um schadhaften Skriptcode einzufügen, der auf den lokalen Browser-Cache eines Benutzers zugreift. Durch die Kombination beider Schwachstellen würde ein Angreifer keinen physischen Zugriff auf das lokale System des Benutzers benötigen.

Empfehlung
Das Speichern von Webseiten innerhalb eines authentifizieren Bereichs einer Anwendung sollte unterbunden werden. Zu diesem Zweck wird empfohlen, die folgenden HTTP-Header für jede Seite zu setzen:

Cache-Control: no-cache, no-store
Pragma: no-cache
Expires: -1

Webapp-04: Cookie ohne "Secure" Eigenschaft

Schaden	Komplexität	Risiko	CVSS:3.1	Status
Mittel	Komplex	Niedrig	CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:U/C:L/I:N/A:N	Offen

Angriffsvoraussetzungen
Der Angreifer ist in der Lage den Netzwerkverkehr zwischen einem Client und Server mitzulesen (Man-in-the-Middle). Zusätzlich muss ein Benutzer die Webanwendung unverschlüsselt über das HTTP-Protokoll aufrufen.

Beschreibung
Bei einer erfolgreichen Anmeldung an der Webanwendung, wird das Sitzungstoken „.ASPXAUTH“ im Cookie ohne die "Secure" Eigenschaft erzeugt:

Set-Cookie: .ASPXAUTH=0F02DB42A1B...; path=/; HttpOnly; SameSite=Abw

Die "Secure" Eigenschaft verhindert, dass Cookies über eine unverschlüsselte HTTP-Verbindung übertragen werden. Aufgrund der Tatsache, dass innerhalb der Webanwendung zusätzlich zur verschlüsselten HTTPS-Verbindung über Port 443 ebenfalls Port 80 für ein unverschlüsseltes HTTP erreichbar ist, kann ein Angreifer dies ausnutzen, um das im Cookie enthaltene Sitzungstoken über einen Man-in-the-Middle Angriff mitzulesen.

Der Angreifer müsste dazu in der Lage sein, den Netzwerkverkehr mitzuschneiden. In diesem Szenario leitet der Angreifer legitime Benutzer der

Webanwendung auf die unverschlüsselte Webseite um. Obwohl die Anwendung eine HTTP-Anfrage automatisch auf eine HTTPS-Anfrage umleitet, würde in dem Moment, wo der Benutzer auf den Link klickt, sein Cookie mit dem Sitzungstoken „.ASPXAUTH“ unverschlüsselt über das Netzwerk übertragen werden und somit von einem Angreifer mitgelesen werden können.

Empfehlung
Die Anwendung sollte die "Secure" Eigenschaft für alle erzeugten Cookies setzen. Das Sitzungstoken würde dann wie folgt erzeugt werden:

Set-Cookie: .ASPXAUTH=0F02DB42A1B...; path=/; HttpOnly; SameSite=Abw; ***Secure***

Webapp-06: Unzureichende Fehlerbehandlung

Schaden	Komplexität	Risiko	CVSS:3.1	Status
Gering	Komplex	Gering	CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:C/C:L/I:N/A:N	Offen

Angriffsvoraussetzungen
Der Angreifer muss die Webanwendung über das Internet erreichen können.

Beschreibung
Die Anwendung gibt im Fehlerfall ausführliche Meldungen (Stack Traces) mitsamt der eingesetzten ASP.NET Version und des .NET Frameworks aus, die teilweise sensible Informationen über die eingesetzte Technologie enthalten. Einem Angreifer könnten diese Informationen für weitere Angriffe gegen die Anwendung oder das zugrundeliegende Framework hilfreich sein.

Beispielsweise kann der Parameter Transaktionen_ClientState im Dashboard der Webanwendung auf den Wert "test" gesetzt werden um folgenden unbehandelten Fehlerfall zu provozieren:

Empfehlung
Fehlerseiten sollten grundsätzlich keine sensiblen Informationen wie technische Interna preisgeben.

Zu diesem Zweck sollten eigene Fehlerseiten implementiert werden, die im Fehlerfall lediglich generische Angaben machen. Weitere Details wie Diagnoseinformationen sollten intern protokolliert und ausgewertet werden.

Webapp-07: Versionsinformationen in HTTP-Headern

Schaden	Komplexität	Risiko	CVSS:3.1	Status
Gering	Moderat	Gering	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N	Offen

Angriffsvoraussetzungen
Der Angreifer benötigt Zugriff auf die Webanwendung.

Beschreibung
Die Webanwendung gibt innerhalb des Headers einer HTTP-Antwort Aufschluss über verwendete Software-Komponenten und Versionen. Dies kann in der folgenden Beispiel-Anfrage gesehen werden:

GET / HTTP/1.1
Host: mandanten-anwendung-test.de

HTTP/1.1 200 OK
Cache-Control: no-cache, no-store
Pragma: no-cache
Content-Type: text/xml; charset=iso-8859-1
Expires: -1
Server: Microsoft-IIS/10.0
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Wed, 25 Oct 2023 07:33:52 GMT
Connection: close
Content-Length: 977

Diese Informationen könnten einem Angreifer dabei helfen, weitere Schwachstellen in den verwendeten Komponenten zu finden und gegen die Webanwendung einzusetzen.

Empfehlung
Alle HTTP-Banner, welche Software-Versionen beinhalten, sollten von der Server-Antwort entfernt werden.

Webapp-09: Veraltete Software in Betrieb

Schaden	Komplexität	Risiko	CVSS:3.1	Status
Mittel	Komplex	Gering	CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N	Offen

Angriffsvoraussetzungen
Der Angreifer benötigt Zugriff auf die Webanwendung.

Beschreibung
Die Anwendung verwendet eine alte Version der JavaScript-Bibliothek jQuery (Version 1.12.4). Diese verfügt über öffentlich bekannte Sicherheitslücken, die, je nach Konfiguration, von einem Angreifer ausgenutzt werden könnten.

Empfehlung:

Die erwähnte Software-Komponente sollte auf die neuste Version aktualisiert werden. Des Weiteren sollten Richtlinien in die Wege geleitet werden, um auch zukünftig Software auf dem neusten Stand zu halten.

Anhang

A. Weitere Informationen

Dieser Bericht ist vertraulich und nur für den internen Gebrauch durch den Empfänger bestimmt.

Für dieses spezielle Projekt wurde eine begrenzte Anzahl von Testtagen verwendet. Die Apollon Security GmbH kann nur während dieses Zeitraums Schwachstellen identifizieren und dokumentieren. Aus diesem Grund kann die Überprüfung in diesem Projekt keinen Anspruch auf Vollständigkeit der in diesem Bericht dokumentierten Sicherheitsschwachstellen erheben. Eine Bewertung des zukünftigen Sicherheitsniveaus oder möglicher zukünftiger Risikoschwachstellen lässt sich aus diesem Bericht nicht ableiten.

Die Penetrationstests wurden professionell durchgeführt und spiegelt den Wissensstand und die Erfahrung der Personen wider, die die Penetrationstests während des Test- und Untersuchungszeitraums durchgeführt haben.

B. Risikobewertung

Folgende Risikostufen, folgend in absteigender Kritikalität, wurden definiert. Diese Farbskala wurde auch für den Report verwendet:

Kritisch
Hoch
Mittel
Gering
Informativ

Eine Risikobewertung erfolgt auf einer Verknüpfung der Schadens- und Komplexitätsbewertung. Dabei wird das Risiko gemessen daran, wie hoch der Schaden bezogen auf Verletzungen der Vertraulichkeit, Integrität und Verfügbarkeit ist.

	Schaden			
Komplexität	Gering	Mittel	Hoch	Kritisch
Trivial				
Einfach				
Moderat				
Komplex				

Apollon Security GmbH bewertet den Schaden eines Angriffs nach deren Auswirkungen auf das Unternehmen. Dabei ist ein Schaden unter anderem ein Imageschaden, ein monetärer Schaden, Personenschaden oder ein Anlagenschaden.

Kritisch	Kritische Auswirkungen auf die Geschäftstätigkeiten des Unternehmens. Es ist von einer starken und andauernden Störung des Betriebs auszugehen. Eine Rufschädigung kann einen großen Personenkreis erreichen, die Vertraulichkeit und Integrität einer Vielzahl von Daten kann nicht mehr sichergestellt werden.
Hoch	Hohe Auswirkungen auf das Unternehmen. Die Verfügbarkeit der Prozesse und Applikationen kann nicht mehr sichergestellt werden. Die Vertraulichkeit und Integrität von einzelnen Daten ist nicht mehr sichergestellt.
Mittel	Es ist von einer kurzen Disruption mit mittlerem Schaden für das Unternehmen zu rechnen. Datendiebstahl ist sehr unwahrscheinlich, eine Rufschädigung kann nur bei einer

	geringen Anzahl von Personen erfolgen.
Gering	Geringe bis minimale Auswirkungen auf das Unternehmen. Es sind nur wenige Mitarbeiter betroffen, wobei die insgesamte Verfügbarkeit des Unternehmens nicht beeinträchtigt ist. Es ist nicht von einer Verletzung der Integrität und Vertraulichkeit zu rechnen. Es ist sehr unwahrscheinlich, dass eine Rufschädigung oder ein Datendiebstahl vorkommt.

Weiterhin wird die Komplexität eines Angriffs wie folgend bewertet. Damit ist der Aufwand gemeint, der zur Ausnutzung einer Schwachstelle benötigt wird. Dieser wird mithilfe des Zeitumfangs, sowie des Wissenstands des Angreifers bemessen:

Trivial	Ein Angriff kann einfach durchgeführt werden, es muss kein weiteres Wissen vorhanden sein
Einfach	Für den Angriff sind Vorkenntnisse und bestimmte Bedingungen (z.B die Nutzung von besonderen Konfigurationen) erforderlich. Die Schwachstelle kann mit wenig Wissen recherchiert und in geringer Zeit ausgenutzt werden. Dabei existieren bereits fertige Werkzeuge zur Ausnutzung dieser Schwachstelle.
Mittel	Die Schwachstelle ist nur ausnutzbar unter speziellen Bedingungen, beispielsweise, wenn Authentisierungsdaten bekannt sind. Außerdem ist ein großer Zeitrahmen, sowie eigene oder neuste Exploits notwendig.
Komplex	Die Schwachstelle ist nur ausnutzbar durch eine Verkettung verschiedener Angriffsmethoden, die nur unter

	speziellen Bedingungen ausnutzbar sind und hohen Aufwand an Vorbereitung benötigen. Hochartige, selbstentwickelte Techniken und Werkzeuge werden benötigt. Es können auch Insiderinformationen notwendig sein.

Apollon Security GmbH
Rheinpromenade 9
40789 Monheim am Rhein

Ihr Kontakt:
kontakt@apollon-security.com