HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
–
Fakultät IV
Wirtschaft und
Informatik

Bedrohungsanalyse

Dennis Grabowski, Julius Zint, Philip Matesanz, Torben Voltmer

Masterprojekt „Entwicklung und Analyse einer sicheren Web-Anwendung“
Wintersemester 18/19

11. November 2018

Inhaltsverzeichnis

1 Zur Analyse verwendeten Methoden

1.1 Datenflussdiagramm

Das folgende Datenflussdiagramm (DFD) wurde mithilfe des Programms „OWASP Threat Dragon“ [2] erstellt. Dieses Programm erlaubt leider keine Einfärbung in grün zur Kennzeichnung vertrauenswürdiger Prozesse oder Datenflüsse. Alle schwarz gekennzeichneten Elemente sind daher als vertrauenswürdig zu betrachten. Dieses DFD soll als Übersicht über alle vorhandenen Datenflüsse dienen, ohne dabei jeden Manager oder Controller seperat aufzulisten, da für diese der Datenfluss der selbe wäre.

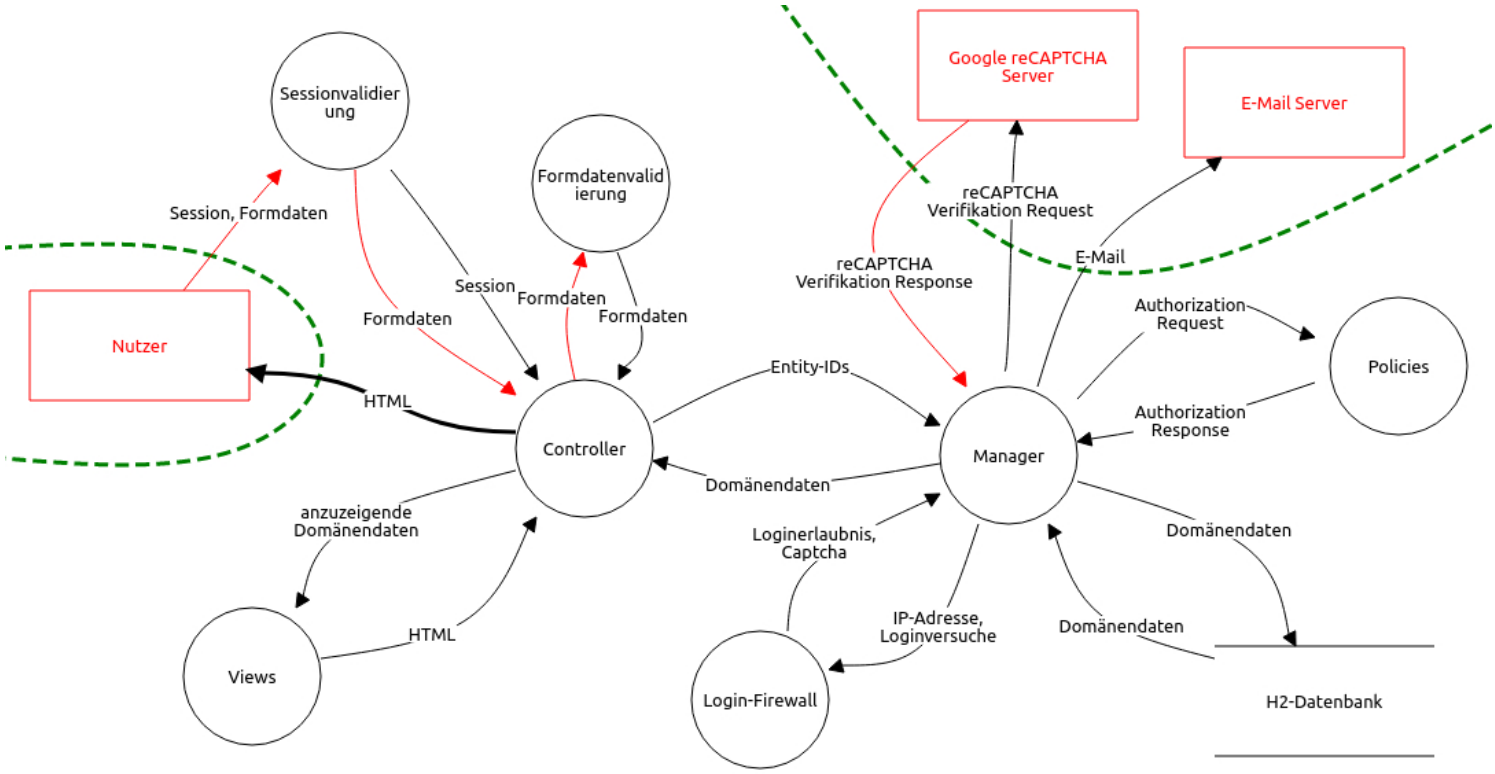

Abbildung 1.1: Gesamtübersicht des Datenflusses in unserer Applikation

1.1.1 Layered Entrypoints

Die folgende Liste stellt alle vorhandenen Eintrittspunkte unserer Applikation in einer Baumstruktur vor. Die Eintrittspunkte wurden für die bessere Übersicht nach `Controller` gruppiert, der für diesen Eintrittspunkt zuständig ist. Zusätzlich werden die Parameter aufgelistet, die wir von dem Nutzer bei dem jeweiligen Eintrittspunkt entgegennehmen. Sie dient hauptsächlich der Vollständigkeit und als Referenzliste, damit kein Eintrittspunkt übersehen werden kann.

1. HTTP (Port 80)
    1.1. `HomeController`
        1.1.1. GET - `/`
    1.2. `LoginController`
        1.2.1. `/login`
            1.2.1.1. GET - `showLoginForm()`
            1.2.1.2. POST - `login(username, password, recaptcha)`
        1.2.2. `/logout`
            1.2.2.1. POST - `logout()`
        1.2.3. `/changePasswordAfterReset`
            1.2.3.1. GET - `showChangePasswordAfterResetForm`
            1.2.3.2. POST - `changePasswordAfterReset(username, currentPassword, password, passwordRepeat, recaptcha)`
    1.3. `UserController`
        1.3.1. `/users`
            1.3.1.1. GET - `showUsers()`
        1.3.2. `/users/create`
            1.3.2.1. GET - `showCreateUserForm()`
            1.3.2.2. POST - `createUser(username, email)`
        1.3.3. `/users/delete`
            1.3.3.1. POST - `deleteUser(userId)`
        1.3.4. `/resetpassword`
            1.3.4.1. GET - `showResetUserPasswordForm`
            1.3.4.2. POST - `resetUserPassword(username)`

1.3.5. `/sessions`

1.3.5.1. GET - `showActiveUserSessions()`

1.3.6. `/sessions/delete`

1.3.6.1. POST - `deleteUserSession()`

1.4. `GroupController`

1.4.1. `/user/groups`

1.4.1.1. GET - `showOwnGroups`

1.4.2. `/groups`

1.4.2.1. GET - `showAllGroups`

1.4.3. `/groups/create`

1.4.3.1. GET - `showCreateGroupForm`

1.4.3.2. POST - `createGroup(groupname)`

1.4.4. `/groups/:groupId`

1.4.4.1. GET - `showGroup(groupId)`

1.4.5. `/groups/:groupId/members/remove`

1.4.5.1. POST - `removeGroupMember(groupId, userId)`

1.4.6. `/groups/:groupId/members/add`

1.4.6.1. POST - `addGroupMember(groupId, userId)`

1.4.7. `/groups/:groupId/delete`

1.4.7.1. POST - `deleteGroup(groupId)`

2. SMTP (Port 587)

2.1. Versand der E-Mail für den Passwort-Reset

2 Bedrohungen

2.1 Gefundene Bedrohungen

2.1.1 Globale Bedrohungen

Unser Schemata zur Bewertung des Risikos einer Bedrohung haben wir teils an die „OWASP Risk Rating Methodology“ angelehnt [1]. Wir verwenden die selbe Unterteilung von Auswirkung und Wahrscheinlichkeit in 3 Stufen, aber bewerten das abschließende Risiko in 4 Stufen „Note“, „Gering“, „Mittel“, „Hoch“.

- **T1: Spoofing des Servers**
  - Zweck:
    - Ausspähen von Anmeldedaten der Benutzer
    - Ausspähen von Dateien, die ein Benutzer beim Hsh-Helfer hochladen möchte
  - Möglichkeiten:
    - IP-Spoofing
    - DNS manipulieren
  - Risiko: Mittel
  - Gegenmaßnahmen: Authentifizierung des Servers durch Zertifikate (HTTPS mit HSTS)
- **T2: Eavesdropping**
  - Zweck:
    - Primär Ausspähen von Anmeldedaten der Benutzer
    - Grundsätzlich sind allen Daten (Gruppennamen etc.) Ziel
  - Möglichkeiten: Man-in-the-Middle-Angriff
  - Risiko: Mittel

[1] Siehe https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology

  - Gegenmaßnahmen: Verschlüsselung durch HTTPS mit HSTS oder Secure-Cookie

- **T3: Replayattacken**
  - Zweck: Mehrfach-Ausführung geschützter Aktionen
  - Möglichkeiten (Bezogen auf unsere Anwendung): Keine
  - Risiko: Note
  - Gegenmaßnahmen: HTTPS
- **T4: Tampering**
  - Zweck:
    * Ausführung von nicht intendierten Aktionen (z.B. Benutzer-Löschen abfangen und Ziel-User-Id manipulieren)
    * Server-Response manipulieren (z.B. bösartiges JavaScript einbauen)
  - Möglichkeiten: Leitung splicen und Man-In-The-Middle durchführen
  - Risiko: Mittel
  - Gegenmaßnahmen: HTTPS
- **T5: Verwendung eines gefälschten,von einer anerkannten Zertifizierungsstelle signierten Zertifikates**
  - Zweck:
    * Spoofing des Servers
    * Man-in-the-Middle-Angriffe
  - Möglichkeiten: Kontrolliert man eine anerkannten Zertifizierungsstelle, können gefälschte Zertifikate für die HsH-Helfer Domain ausgestellt werden. Für HsH-Helfer Benutzer ist das nicht sofort einsehbar[2].
  - Risiko: Gering
  - Gegenmaßnahmen: HTTP Public Key Pinning (HPKP)
- **T6: XSS (Cross Site Scripting)**
  - Zweck: Ausführung von JavaScript im HsH-Helfer Kontext eines Benutzers
  - Möglichkeiten: Eingaben, die ausgegeben werden, ohne Eingabevalidierung oder Ausgabecodierung auszuführen

[2]Bereits geschehen durch türkischen Geheimdienst: https://arstechnica.com/information-technology/2013/01/turkish-government-agency-spoofed-google-certificate-accidentally/

  - Risiko: Hoch
  - Gegenmaßnahmen:
    * Eingabevalidierung (Kleinbuchstaben und Zahlen, valide E-Mail-Adressen)
    * X-XSS-Protection (vgl. 2.3.4)
    * Content Security Policy (CSP) des Browsers aktivieren
    * Alte Browser bzw Browser, die CSP nicht können, aussperren (s. Zusatzfrage)
    * Ausgabecodierung

- **T7: SQL Injection**
  - Zweck: Unzulässiges Lesen und Schreiben von Datenbankinhalten
  - Möglichkeiten: Inline-SQL Queries ohne Seperation of Data and Code
  - Risiko: Hoch
  - Gegenmaßnahmen:
    * Die H2 Einstellung `ALLOW_LITERALS=NONE` verwenden
    * Nur PreparedStatements benutzen
    * Eingabevalidierung

- **T8: Offline Password Brute Forcing**
  - Zweck: Durch SQL Injection erbeutete Hashes in Plaintext verwandeln
  - Möglichkeiten: Rainbowtables
  - Risiko: Gering
  - Gegenmaßnahmen:
    * Passwörter nur gehashed mit Salt speichern
    * Passwörter zusätzlich mit Pepper hashen

- **T9: Cross-Site-Request-Forgery (CSRF)**
  - Zweck: Benutzer dazu bringen Aktionen auszuführen, die sie nicht intendiert haben
  - Möglichkeiten:
    * Einen fremden Benutzer dazu bringen, sich selbst auszuloggen
    * Einem Administrator einen Request unterjubeln, durch welchen er einen Nutzer erstellt

  - Risiko: Hoch
  - Gegenmaßnahmen: CSRF-Tokens bei Requests prüfen
- **T10: Session fälschen (Bestimmte oder Irgendeine)**
  - Zweck: Client spoofen um Aktionen auszuführen, für die der gespoofte Client autorisiert ist
  - Möglichkeiten: Cookie fälschen
  - Risiko: Hoch
  - Gegenmaßnahmen:
    * Signieren des Cookies durch privaten Schlüssel,
    * Verwendung von UUID,
    * Bindung der Session an IP-Adresse,
    * Begrenzung der Lebenszeiten
- **T11: Abstreitbarkeit illegaler Handlungen**
  - Zweck: Anderen Nutzer in rechtliche Schwierigkeiten bringen
  - Möglichkeiten: Eintragen illegaler Namen bei Benutzernamen oder Gruppenmane
  - Risiko: Gering
  - Gegenmaßnahmen: Logging von IP + Zeit, Action, User
- **T12: Nutzer kann eine ihm unerlaubte Operation durchführen**
  - Zweck: Beispielsweise um einen Denial-of-Service-Angriff auf einen Nutzer durchzuführen
  - Möglichkeiten: Beispielsweise Nutzer löscht andere Nutzer
  - Risiko: Mittelmäßig - Hoch
  - Gegenmaßnahmen: Autorisierung erzwingen und prüfen bei jeder Operation
- **T13: Captchas durch menschliche Akteure lösen lassen**
  - Zweck: Brute-Forcing ermöglichen
  - Möglichkeiten: 2captcha.com bietet diesen Service für 2.99$ für 1000 reCAPTCHAs
  - Risiko: Gering
  - Gegenmaßnahmen: Zwei-Faktor-Authentisierung
- **T14: Distributed-Denial-of-Service**

  - Zweck: Verfügbarkeit der Anwendung beeinträchtigen
  - Möglichkeiten: Botnetzwerk oder Booter-Service verwenden
  - Risiko: Mittel
  - Gegenmaßnahmen: Cloudflare/Incapsula/Akamai Secure Shield verwenden

2.1.2 Domänenspezifische Bedrohungen

Mögliche Angriffe auf den Loginprozess

- **T15: Überlanges Passwort beim Login eingeben**
  - Zweck: DOS der Applikation
  - Möglichkeiten: Durch überlange Passwörter eine ressourcenintensive Hash-Operation in Gang setzen
  - Risiko: Hoch
  - Gegenmaßnahmen:
    * Passwort-Länge mittels Eingabevalidierung beschränken
    * Verwendung von bcrypt (lässt nur 72 Zeichen lange Passwörter zu, Laufzeit ist für alle Eingabelängen konstant)
- **T16: Timing-Angriff auf Loginverhalten**
  - Zweck: Herausfinden, ob ein Benutzer mit einem bestimmten Username existiert
  - Möglichkeiten: Durch Messen der Antwortszeiten des Servers können Rückschlüsse darüber gezogen werden, welcher Code ausgeführt wurde (zB Hashing/DB-Zugriff)
  - Risiko: Mittel
  - Gegenmaßnahmen: Loginprozess für jeden Fall gleich ablaufen lassen
- **T17: Online Brute-Forcing**
  - Zweck: Unautorisierter Zugriff auf ein fremdes Benutzerkonto
  - Möglichkeiten: Beispielsweise durch Wörterbuchattacken
  - Risiko: Hoch
  - Gegenmaßnahmen:
    * Login nur möglich, wenn Captcha gelöst wird

    * IP Sperre bei auffälligem Verhalten
    * Schwache Passwörter verbieten
    * Zwei-Faktor-Authentisierung

Mögliche Angriffe auf den „Passwort zurücksetzen lassen“-Prozess

- **T18: Nutzer permanent aus der Applikation ausschließen**
  - Zweck: Denial-of-Service-Angriff auf einen individuellen Nutzer
  - Möglichkeiten: Wiederholtes Absenden des Request mit richtigen Nutzernamen generiert bei jedem Request ein neues temporäres Passwort. Selbst wenn der Benutzer das temporäre Passwort ändert, wird er es nicht schaffen sich anzumelden, da dann bereits die Generierung ein neues temporäres Passwort vom Angreifer erzwungen wurde.
  - Risiko: Hoch
  - Gegenmaßnahmen:
    * Einbau eines Captchas
    * Dem Benutzer einen Link schicken, der geöffnet werden muss, um den „Passwort ändern“-Prozess anzustoßen. Der Link muss einen geheimen Token beinhalten, der nur HsH-Helfer und dem Empfänger der E-Mail bekannt ist. Das temporäre Passwort wird erst generiert, sobald auf den Link geklickt wird.
- **T19: Timing-Angriff auf Verhalten des Requests**
  - Zweck: Herausfinden, ob ein Benutzer mit einem bestimmten Username existiert
  - Möglichkeiten: Durch Messen der Antwortszeiten des Servers können Rückschlüsse darüber gezogen werden, welcher Code ausgeführt wurde (E-Mail-Versand)
  - Risiko: Mittel
  - Gegenmaßnahmen:
    * „Passwort zurücksetzen lassen“-Prozess in konstanter Länge ablaufen lassen, ggf. durch Absenden einer E-Mail an eine Dummy-E-Mail-Adresse
    * E-Mail-Versand asynchron durchführen
- **T20: E-Mail Postfach des Nutzers fluten**
  - Zweck: Verbrauch der vorhandenen Resourcen seines E-Mail-Kontos

- Möglichkeiten: Wiederholtes Absenden des Formulars durch Angabe seines Nutzernamens
- Risiko: Hoch
- Gegenmaßnahmen: Nutzer muss Captcha lösen, bevor Prozess angestoßen werden darf

Mögliche Angriffe auf den „Passwort ändern“-Prozess

Da dieser Prozess dem Loginprozess gleicht, treten hier die selben Lücken auf. Um das Dokument kurz zu halten, listen wir diese nicht nochmal auf.

Mögliche Angriffe beim Versand von E-Mails

Der Angriffsvektor SMTP sollte ausweislich der Aufgabenstellung nicht betrachtet werden. Wir listen aus diesem Grund keine diesbezüglichen Bedrohungen auf.

2.2 Ignorierte Bedrohungen

2.2.1 Captchas durch menschliche Akteure lösen lassen

Die Kosten für diese Dienstleistung machen sie unwirtschaftlich in Anbetracht unseres Applikationskontexts.

2.2.2 HTTP bezogene Bedrohungen

Die Bedrohungen T1, T2, T3, T4, T5 werden ignoriert. Ausweislich der Aufgabenstellung sollen wir eine sichere Verbindung annehmen.

2.3 Vorzunehmende Gegenmaßnahmen

In der Tabelle 2.1 können die Gegenmaßnahmen, die wir geplant haben umzusetzen, entnommen werden.

Tabelle 2.1: Auflistung aller vorzunehmenden Gegenmaßnahmen mit Referenz auf Angriff, der verhindert/erschwert wird. Status beschreibt bereits durchgeführte Implementation.

Gegenmaßnahme	**Angriff**	**Status**
Eingabevalidierung	T6, T7, T15	✓
Ausgabecodierung	T6	✓
PreparedStatements	T7	✓
`ALLOW_LITERALS=NONE` aktivieren	T7	✓
Salt auf Passwörter anwenden	T8	✓
Pepper auf Passwörter anwenden	T8	✗
Hinzufügen von CSRF-Token zu Formularen/Requests	T9	✓
Cookies mit privaten Schlüssel signieren	T10	✓
Session-ID im UUID-Format persistieren	T10	✓
Session an IP-Adresse binden	T10	✓
Session-Lebenszeit stark begrenzen	T10	✓
Aktionen inkl. User, IP-Adresse und Zeit loggen	T11	✗
Zwei-Faktor-Authentisierung	T13, T17	✗
bcrypt als Hashing-Algorithmus verwenden	T15	✓
Operationen unter konstanter, gleichbleibender Laufzeit - auch bei unterschiedlichen Ausführungssträngen - ausführen	T16, T19	✓
Verbieten schwacher Passwörter	T17	✗
Captcha zwischenschalten, um Massenrequests entgegenzuwirken	T17, T18, T20	✓
IP-Adresse bei auffälligem Verhalten sperren	T17	✓
Aussperrung alter Browserversionen	T6	✗
„Content Security Policy“-Header nutzen	T6	✓
„X-XSS-Protection“-Header nutzen	T6	✓

2.3.1 HTTP Header

In der folgenden Liste werden verschiedene sicherheitsrelevante „HTTP Headers“ beschrieben, die uns helfen, einige der Gegenmaßnahmen zu realisieren. Diese Werte werden empfohlen, um die „maximale“ Sicherheit zu garantieren. Entnommen haben wir diese Empfehlungen verschiedenen Seiten, darunter der „Mozilla's Enterprise Information Security“-Blog [3] und das „OWASP Secure Headers Project“ [1], die wir beide als vertrauenswürdig betrachten.

2.3.2 Referrer-Policy

```
Referrer-Policy: origin-when-cross-origin, strict-origin-when-cross-origin
```

Mit diesem Header kann gesteuert werden, welche Informationen als „Referrer“ an die nächste besuchte Seite weitergegeben werden. `strict-origin-when-cross-origin` sorgt dafür, dass innerhalb der Seite die vollständige URL weitergegeben wird. Beim Besuchen einer anderen Seite wird nur die Domain weitergegeben. Wird dabei jedoch von HTTPS auf HTTP gewechselt, ist das Weitergeben jeder Information untersagt.

2.3.3 X-Frame-Options

```
X-Frame-Options: DENY
```

Um zu verhindern, dass unsere Seite in einem `<frame>`, `<iframe>` oder `<object>` Element auf einer anderen Seite eingebettet wird, kann der Header `X-Frame-Options` verwendet werden. So kann Clickjacking verhindert werden.

2.3.4 X-XSS-Protection

```
X-XSS-Protection: 1; mode=block
```

Durch den Header `X-XSS-Protection` wird der Browser angewiesen, eine XSS Erkennung zu versuchen. Dieses Feature ist für älteren Browser interessant, die noch kein CSP implementieren. Durch den Modus „block“ wird das Laden der Seite abgebrochen, sobald ein XSS Versuch erkannt wird.

2.3.5 X-Content-Type-Options

```
X-Content-Type-Options: nosniff
```

Mit diesem Header wird dem Browser verboten, „MIME sniffing“[3] zu betreiben. Er wird gezwungen den MIME Typ zu verwenden, der im `Content-Type` Header angegeben wurde.

2.3.6 Content-Security-Policy

```
Content-Security-Policy: default-src 'self'; script-src
https://www.google.com/recaptcha/ https://www.gstatic.com/recaptcha/;
frame-src https://www.google.com/recaptcha/
```

Innerhalb diesem Header ist es möglich Richtlinien zu definieren, die den Zugriff auf verschiedene Resourcen kontrollieren. Dadurch kann verhindert werden, dass JavaScript

[3] Beim „MIME sniffing“ versucht der Browser den richtigen MIME Type durch Betrachtung des Inhalts zu erraten, wenn er annimmt, dass der im `Content-Type` übermittelte Wert nicht korrekt ist.

nicht von jeder URL geladen und ausgeführt werden darf, von welchen Quellen Bilder geladen werden dürfen, oder welches CSS eingebunden werden darf. Dieser Header ist dem „X-XSS-Protection“-Header vorzuziehen.

3 Literatur

[1] *OWASP Secure Headers Project*. URL: https://www.owasp.org/index.php/OWASP_Secure_Headers_Project (besucht am 10. 11. 2018).

[2] *OWASP Threat Dragon Project - Threat Modelling Application*. URL: https://www.owasp.org/index.php/OWASP_Threat_Dragon (besucht am 10. 11. 2018).

[3] *Web Security*. URL: https://infosec.mozilla.org/guidelines/web_security#cross-origin-resource-sharing (besucht am 10. 11. 2018).

Abkürzungsverzeichnis

DFD Datenflussdiagramm 3, *Glossareintrag:* Datenflussdiagramm

JWT JSON Web Token *Glossareintrag:* JSON Web Token

UUID Universally Unique Identifier *Glossareintrag:* Universally Unique Identifier

Glossar

Datenflussdiagramm Ein Datenflussdiagramm zeigt über festgelegte Notation auf, welche Daten von welchem Akteur über welchen Eintrittspunkt bei welchem Prozess des Systems ankommen. 3

Abbildungsverzeichnis