

Bedrohungsanalyse

Dennis Grabowski, Julius Zint, Philip Matesanz, Torben Voltmer

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

28. November 2018



Inhaltsverzeichnis

1	Zur Analyse verwendeten Methoden	3
1.1	Datenflussdiagramm	3
1.1.1	Layered Entrypoints	4
2	Bedrohungen	9
2.1	Gefundene Bedrohungen	9
2.1.1	Globale Bedrohungen	9
2.1.2	Domänenspezifische Bedrohungen	13
2.1.3	Mögliche Angriffe auf den „Datei hochladen“-Prozess	15
2.1.4	Mögliche Angriffe auf den „Dateien anzeigen lassen“-Prozess	15
2.1.5	Mögliche Angriffe auf den „Auf Datei zugreifen“-Prozess	16
2.2	Ignorierte Bedrohungen	19
2.2.1	Captchas durch menschliche Akteure lösen lassen	19
2.2.2	HTTP bezogene Bedrohungen	19
2.3	Vorzunehmende Gegenmaßnahmen	19
2.3.1	HTTP Header	20
2.3.2	Referrer-Policy	20
2.3.3	X-Frame-Options	20
2.3.4	X-XSS-Protection	20
2.3.5	X-Content-Type-Options	21
2.3.6	Content-Security-Policy	21
	Abbildungsverzeichnis	23

1 Zur Analyse verwendeten Methoden

1.1 Datenflussdiagramm

Das folgende **Datenflussdiagramm (DFD)** wurde mithilfe des Programms „OWASP Threat Dragon“ [OWASP.ThreatDragon] 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 separat aufzulisten, da für diese der Datenfluss der selbe wäre.

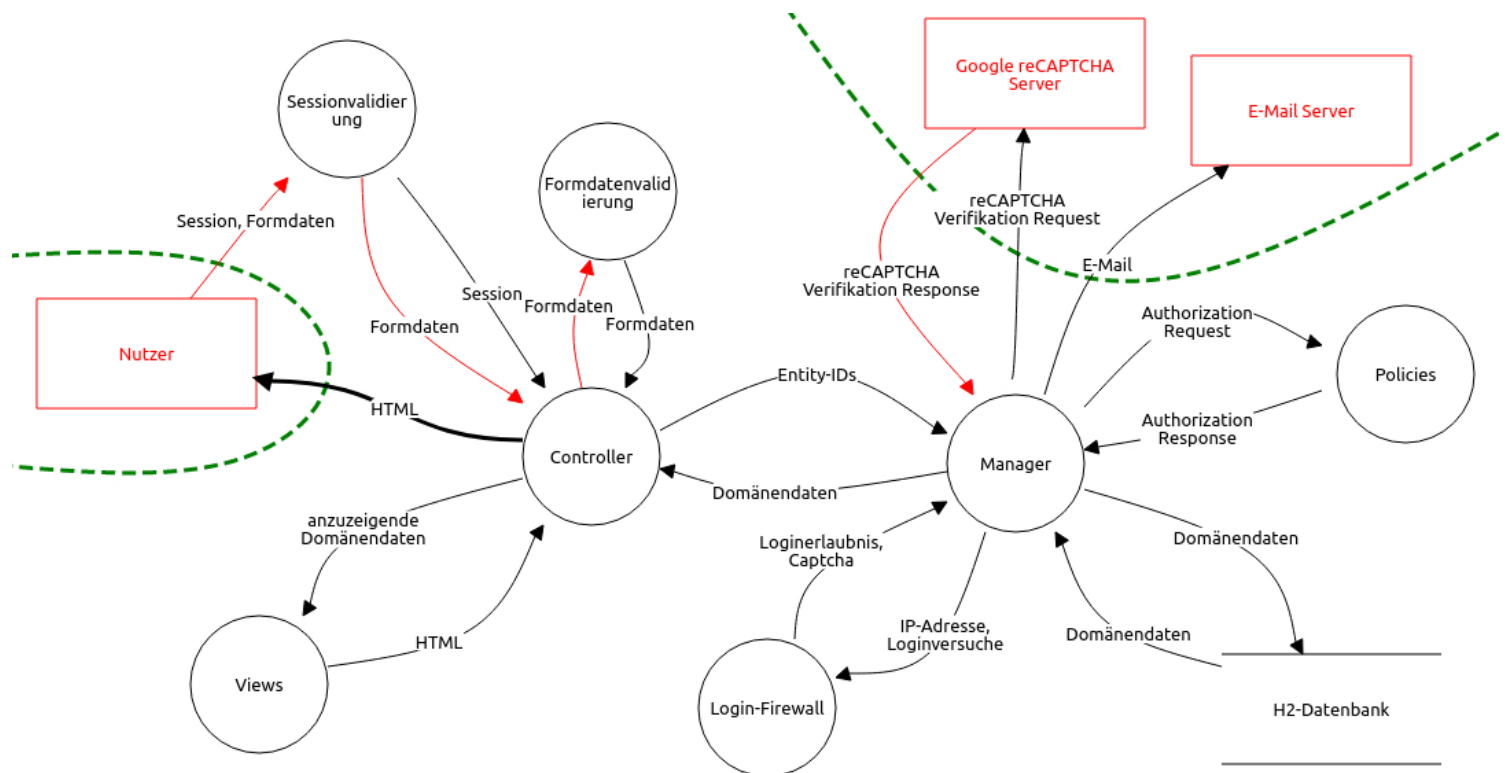


Abbildung 1.1: Gesamtübersicht des Datenflusses in unserer Applikation

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. Um diese kurz und übersichtlich zu halten, geben wir nicht die Session an, die ein Nutzer implizit mitschickt. Sie dient hauptsächlich der Vollständigkeit und als Referenzliste, damit kein Eintrittspunkt übersehen werden kann.

1. HTTP (Port 80)

1.1. HomeController

1.1.1. [/](#)

1.1.1.1. GET - [/index\(\)](#)

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/all](#)

1.3.1.1. GET - `showUsers()`

1.3.2. [/users/admins](#)

1.3.2.1. [GET - showAdminUsers\(\)](#)

1.3.3. [/users/create](#)

1.3.3.1. GET - `showCreateUserForm()`

1.3.3.2. POST - `createUser(username, email, quotaLimit)`

1.3.4. [/users/confirmDelete](#)

- 1.3.4.1. [GET - showConfirmDeleteForm\(userId\)](#)
- 1.3.5. [/users/delete](#)
 - 1.3.5.1. POST - deleteUser(userId)
- 1.3.6. [/users/:userId/settings](#)
 - 1.3.6.1. [GET - showUserAdminSettings\(userId\)](#)
- 1.3.7. [/users/editQuota](#)
 - 1.3.7.1. [GET - changeUserQuotaLimit\(userId, newQuotaLimit\)](#)
- 1.3.8. [/resetpassword](#)
 - 1.3.8.1. GET - showResetUserPasswordForm
 - 1.3.8.2. POST - resetUserPassword(username, [recaptcha](#))
- 1.3.9. [/sessions](#)
 - 1.3.9.1. GET - showActiveUserSessions([userId](#))
- 1.3.10. [/sessions/changePassword](#)
 - 1.3.10.1. [POST - changeUserPassword\(\)](#)
- 1.3.11. [/sessions/delete](#)
 - 1.3.11.1. POST - deleteUserSession([sessionId](#))
- 1.3.12. [/settings](#)
 - 1.3.12.1. [POST - showUserSettings\(\)](#)
- 1.3.13. [/settings/changePassword](#)
 - 1.3.13.1. [POST - changeUserPassword\(currentPassword, newPassword\)](#)
- 1.3.14. [/settings/sessionTimeout](#)
 - 1.3.14.1. [POST - changeUserSessionTimeout\(newSessionTimeout\)](#)
- 1.4. GroupController
 - 1.4.1. [/user/groups/all](#)
 - 1.4.1.1. GET - [showOwnGroupsshowAllGroups\(\)](#)
 - 1.4.2. [/groups/membership](#)
 - 1.4.2.1. GET - [showAllGroupsshowOwnMemberships\(\)](#)
 - 1.4.3. [/groups/own](#)
 - 1.4.3.1. [GET - showOwnGroups\(\)](#)

- 1.4.4. [/groups/delete](#)
 - 1.4.4.1. [POST - deleteGroup\(groupId\)](#)
- 1.4.5. [/groups/confirmDelete](#)
 - 1.4.5.1. [POST - confirmDelete\(groupId\)](#)
- 1.4.6. [/groups/create](#)
 - 1.4.6.1. [GET - showCreateGroupForm\(\)](#)
 - 1.4.6.2. [POST - createGroup\(groupnamegroupName\)](#)
- 1.4.7. [/groups/:groupId](#)
 - 1.4.7.1. [GET - showGroup\(groupId\)](#)
- 1.4.8. [/groups/:groupId/files](#)
 - 1.4.8.1. [GET - showGroupFiles\(groupId\)](#)
- 1.4.9. [/groups/:groupId/members](#)
 - 1.4.9.1. [GET - showGroupMembers\(groupId\)](#)
- 1.4.10. [/groups/:groupId/members/remove](#)
 - 1.4.10.1. [POST - removeGroupMember\(groupId, userId\)](#)
- 1.4.11. [/groups/:groupId/members/add](#)
 - 1.4.11.1. [GET - showAddMemberForm\(groupId\)](#)
 - 1.4.11.2. [POST - addGroupMember\(groupId, userId\)](#)
- 1.5. [/groups/:groupId/deleteFileController](#)
 - 1.5.1. [/files/own](#)
 - 1.5.1.1. [GET - showOwnFiles\(\)](#)
 - 1.5.2. [/files/shared](#)
 - 1.5.2.1. [GET - showSharedFiles\(\)](#)
 - 1.5.3. [/files/thirdParty](#)
 - 1.5.3.1. [GET - showThirdPartyFiles\(\)](#)
 - 1.5.4. [/files/delete](#)
 - 1.5.4.1. [POST - deleteGroupdeleteFile\(groupIdfileId\)](#)
 - 1.5.5. [/files/upload](#)
 - 1.5.5.1. [GET - showUploadFileForm\(\)](#)

- 1.5.5.2. [POST - uploadFile\(fileName, fileContent, comment, groupPermissions, userPermissions\)](#)
- 1.5.6. [/files/uploadToGroup/:groupId](#)
 - 1.5.6.1. [GET - showUploadFileToGroupForm\(groupId\)](#)
- 1.5.7. [/files/quota](#)
 - 1.5.7.1. [GET - showQuotaUsage\(\)](#)
- 1.5.8. [/files/:fileId](#)
 - 1.5.8.1. [GET - showFile\(fileId\)](#)
- 1.5.9. [/files/:fileId/download](#)
 - 1.5.9.1. [GET - downloadFile\(fileId\)](#)
- 1.5.10. [/files/editComment](#)
 - 1.5.10.1. [POST - editFileComment\(fileId, comment\)](#)
- 1.5.11. [/files/editContent](#)
 - 1.5.11.1. [POST - editFileContent\(fileId, newFileContent\)](#)
- 1.5.12. [/files/search](#)
 - 1.5.12.1. [POST - searchFiles\(query\)](#)
- 1.6. [PermissionController](#)
 - 1.6.1. [/permissions/editUserPermission](#)
 - 1.6.1.1. [POST - showEditUserPermissionForm\(userPermissionsId\)](#)
 - 1.6.2. [/permissions/editUserPermission/submit](#)
 - 1.6.2.1. [POST - editUserPermission\(userPermissionId, permissionLevel, returnUrl\)](#)
 - 1.6.3. [/permissions/editGroupPermission/](#)
 - 1.6.3.1. [POST - showEditGroupPermissionForm\(userPermissionId, returnUrl\)](#)
 - 1.6.4. [/permissions/editGroupPermission/submit](#)
 - 1.6.4.1. [POST - editGroupPermission\(groupPermissionId, permissionLevel, returnUrl\)](#)
 - 1.6.5. [/permissions/createUserPermission/:fileId](#)
 - 1.6.5.1. [GET - showCreateUserPermission\(fileId\)](#)
 - 1.6.6. [/permissions/createUserPermission](#)

- 1.6.6.1. [POST - createUserPermission\(fileId, userId, permissionLevel\)](#)
 - 1.6.7. [/permissions/createGroupPermission/:fileId](#)
 - 1.6.7.1. [GET - showCreateGroupPermission\(fileId\)](#)
 - 1.6.8. [/permissions/createGroupPermission/](#)
 - 1.6.8.1. [POST - createGroupPermission\(fileId, groupId, permissionLevel\)](#)
 - 1.6.9. [/permissions/deleteGroupPermission](#)
 - 1.6.9.1. [POST - deleteGroupPermission\(groupPermissionId, returnUrl\)](#)
 - 1.6.10. [/permissions/deleteUserPermission](#)
 - 1.6.10.1. [POST - deleteUserPermission\(userPermissionId, returnUrl\)](#)
 - 1.6.11. [/permissions/editGroupPermission/submit](#)
 - 1.6.11.1. [POST - editGroupPermission\(\)](#)
 - 1.7. [ErrorController](#)
 - 1.7.1. [/forbidden](#)
 - 1.7.1.1. [GET - showForbiddenMessage\(\)](#)
 - 1.7.2. [/badrequest](#)
 - 1.7.2.1. [GET - showBadRequestMessage\(\)](#)
 - 1.8. [/assets/*file](#)
 - 1.8.1. [GET - Assets.at\(path="/public", file\)](#)
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
- Mögliche 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

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

- Mögliche Gegenmaßnahmen: Verschlüsselung durch HTTPS mit HSTS oder SecureCookie
- **T3: Replayattacken**
 - Zweck: Mehrfach-Ausführung geschützter Aktionen
 - Möglichkeiten (Bezogen auf unsere Anwendung): Keine
 - Risiko: Note
 - Mögliche 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. böses JavaScript einbauen)
 - Möglichkeiten: Leitung splicen und Man-In-The-Middle durchführen
 - Risiko: Mittel
 - Mögliche 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 einsehbar².
 - Risiko: Gering
 - Mögliche 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 Eingabvalidierung oder Ausgabecodierung auszuführen

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

- Risiko: Hoch
- Mögliche 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 Separation of Data and Code
 - Risiko: Hoch
 - Mögliche 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
 - Mögliche 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
- [Mögliche](#) 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
 - [Mögliche](#) 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
 - [Mögliche](#) 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
 - [Mögliche](#) 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
 - [Mögliche](#) Gegenmaßnahmen: Zwei-Faktor-Authentisierung

- **T14: Distributed-Denial-of-Service**

- Zweck: Verfügbarkeit der Anwendung beeinträchtigen
- Möglichkeiten: Botnetzwerk oder Booter-Service verwenden
- Risiko: Mittel
- Mögliche 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
- Mögliche 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 Antwortzeiten des Servers können Rückschlüsse darüber gezogen werden, welcher Code ausgeführt wurde (zB Hashing/DB-Zugriff)
- Risiko: Mittel
- Mögliche 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

- Mögliche 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
- Mögliche Gegenmaßnahmen:
 - * Einbau eines Captchas
 - * Dem Benutzer einen Link schicken, der geöffnet werden muss, um den „Passwort nach Zurücksetzung 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 Antwortzeiten des Servers können Rückschlüsse darüber gezogen werden, welcher Code ausgeführt wurde (E-Mail-Versand)
- Risiko: Mittel
- Mögliche 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 Ressourcen seines E-Mail-Kontos
- Möglichkeiten: Wiederholtes Absenden des Formulars durch Angabe seines Nutzernamens
- Risiko: Hoch
- Mögliche Gegenmaßnahmen: Nutzer muss Captcha lösen, bevor Prozess angestoßen werden darf

Mögliche Angriffe auf den „Passwort nach Zurücksetzung 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.

2.1.3 Mögliche Angriffe auf den „Datei hochladen“-Prozess

- **T21: Dateinamen oder Kommentarfeld zum Dateien kodieren missbrauchen**

- Zweck: Speicherplatzlimit umgehen, Speicherverbrauch des Servers hochtreiben
- Möglichkeiten: Dateiinhalt in Kommentarfeld oder Dateinamen schreiben, gegebenenfalls sogar kodiert.
- Risiko: Mittel
- Mögliche Gegenmaßnahmen: Kommentarfeldlänge und Dateinamenlänge zählen ins Speicherplatzlimit des Nutzers

2.1.4 Mögliche Angriffe auf den „Dateien anzeigen lassen“-Prozess

- **T22: Irreführung des Nutzers durch ähnliche Dateinamen**

- Zweck: Nutzer eine bösartige Datei unterjubeln
- Möglichkeiten: Datei mit gleichem Dateinamen hochladen, wie besagter Nutzer bereits hochgeladen hat und ihm Lese- und Schreibrechte geben
- Risiko: Hoch
- Mögliche Gegenmaßnahmen: Hervorhebung der Nutzer-eigenen Datei, beispielsweise durch Farben oder Symbolen

2.1.5 Mögliche Angriffe auf den „Auf Datei zugreifen“-Prozess

- T23: Dateien hochladen, die vom Browser interpretiert werden
 - Zweck: Code-Injection, Cookies klauen
 - Möglichkeiten: Cookies klauen durch JavaScript-Injektion
 - Risiko: Kritisch
 - Mögliche Gegenmaßnahmen:
 - * HTTP-Header Content-Type auf application/octet-stream setzen
 - * User-generierte Inhalte über eine andere Domain ausliefern
 - * Dateien beim Download zippen
- T24: Dateiname erlaubt HTTP Response Splitting
 - Zweck: Schadcode injizieren
 - Möglichkeiten: Doppelte Newlines im Dateinamen
 - Risiko: Hoch
 - Mögliche Gegenmaßnahmen: Eingabevalidierung, damit doppelte Newlines nicht mehr erlaubt sind

Mögliche Angriffe auf Zusatzfunktionalität: Datei-Berechtigungen editieren

- T25: Open Redirect
 - Zweck: Benutzer auf eine Angriffsseite umleiten.
 - Möglichkeiten: Eigene URL als Redirect-Parameter angeben.
 - Risiko: Mittel
 - Mögliche Gegenmaßnahmen:
 - * Zugrundeliegende Funktionalität nicht per GET sondern per POST anbieten, das vom Play CSRF-Schutz gesichert wird.
 - * URL-Eingabe durch ein Regex prüfen.
 - * URL-Eingabe durch eine Whitelist prüfen.

Mögliche Angriffe auf das Logging

- T26: Log Poisoning
 - Zweck: Log-Einträge für real nicht stattgefunden Events erstellen / Anderem Nutzer etwas anhängen.
 - Möglichkeiten: User-Input in Logs mit Newline versehen.
 - Risiko: Mittel
 - Mögliche Gegenmaßnahmen: Ausgabecodierung von User-Input in Logdatei z.B. Base64.
- T27: Log Code Injection
 - Zweck: Programme angreifen, welche die Log parsen/öffnen.
 - Möglichkeiten: Schadcode in User-Input eingeben, der in der Log-Datei landet.
 - Risiko: Low
 - Mögliche Gegenmaßnahmen: - Wird derzeit noch diskutiert -
- T28: Speicherverbrauch durch Spammen von Requests
 - Zweck: Speicher des Servers völlig mit Logs belegen -j. Denial of Service.
 - Möglichkeiten: In endlosschleife Requests ausführen.
 - Risiko: Mittel
 - Mögliche Gegenmaßnahmen: Log-Rotation verwenden.
- T29: Log-Deletion durch Spammen von Requests
 - Zweck: Log-Rotation triggern, um eine Beweisvernichtung auszuführen.
 - Möglichkeiten: In endlosschleife Requests ausführen.
 - Risiko: Mittel
 - Mögliche Gegenmaßnahmen: - Wird derzeit noch diskutiert -
- T30: Virensignatur in User-Input
 - Zweck: Denial of Service / Logging blockieren
 - Möglichkeiten: Eine Signatur in User-Input einbauen, die von einem Virenscanner erkannt wird. Virenscanner blockiert folgend Zugriffe auf die Log-Datei und Anwendung kann keine weiteren Logs schreiben.
 - Risiko: Low

- Mögliche Gegenmaßnahmen:
 - * Keinen Virens Scanner auf Applikationsserver verwenden.
 - * Anwendung in VM ohne Virens Scanner laufen lassen.
 - * Log-Folder auf Whitelist des Virens Scanner setzen.

Mögliche Angriffe auf Zusatzfunktionalität: Anzeige, wer File als letztes beschrieben haben

- T31: CSS Injection
 - Zweck: Einfärbewarnung deaktivieren.
 - Möglichkeiten: Wie XSS, jedoch wird anstelle von Javascript Inline-CSS injiziert, das die CSS-Klasse überschreibt, welche für die Rotfärbung von Dateien verantwortlich ist.
 - Risiko: Mittel
 - Mögliche Gegenmaßnahmen:
 - * Ausgabecodierung.
 - * Content-Security-Policy.

Mögliche Angriffe auf Zusatzfunktionalität: Nutzer kann Session-Timeout einstellen

- T32: Überlanges Session-Timeout einstellen
 - Zweck: Umgehen des intendierten Schutzmechanismus (Session-Timeout).
 - Möglichkeiten: Einen extrem hohen Wert wählen.
 - Risiko: Mittel
 - Mögliche Gegenmaßnahmen:
 - * Einen Maximalwert definieren.
- T33: Integer Overflow
 - Zweck: Umgehen des intendierten Schutzmechanismus (Session-Timeout).
 - Möglichkeiten: Einen extrem hohen Wert wählen.
 - Risiko: Mittel
 - Mögliche Gegenmaßnahmen:

- * Einen Maximalwert definieren.
- * Arithmetik verwenden, die nicht anfällig für Integer-Overflows ist.
- **T34: Negatives Session Timeout**
 - Zweck: Umgehen des intendierten Schutzmechanismus (Session-Timeout).
 - Möglichkeiten: Einen sehr geringen (negativen) Wert wählen.
 - Risiko: Mittel
 - Mögliche Gegenmaßnahmen:
 - * Einen Minimalwert definieren.

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 entnommen werden, die wir **geplant haben umzusetzen, entnommen werden** planen umzusetzen.

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 [[Mozilla.SecureHeaders](#)] und das „OWASP Secure Headers Project“ [[OWASP.SecureHeaders](#)], 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 ([Chrome](#), [Safari](#), [Internet Explorer](#) und [Opera](#)) angewiesen, eine ~~XSS-Erkennung~~ „reflected 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.~~ Dieser Header dient daher als zusätzliches Auffangnetz, im Fachjargon oftmals „Defense in Depth“ oder „castle approach“ genannt, aber nicht als ausreichender Schutz. Leider ist dies relativ einfach auszuhebeln oder sogar zu missbrauchen, daher fordern wir in unseren Annahmen explizit ein Ausschluss älterer Browser. Die oben genannten Browser verwenden hierfür ein Blacklisting verschiedener Tags sowie gefährlicher Zeichen, bei welchen der „XSS-Auditor“ anspringt und die

potentielle XSS-Payload herausfiltert.³ Solch ein Blacklisting ist allerdings aufgrund verschiedener Möglichkeiten zum Deformieren des Payloads einfach auszuhebeln. Darüber hinaus kann der XSS-Auditor auch missbraucht werden, um bestimmte JavaScript-Bibliotheken von einer Seite zu entfernen. Chrome's/Chromium's XSS-Auditor⁴ entfernt beispielsweise `<script>`-Tags in dem HTML, wenn sie zeitgleich im Request-Query-String gefunden werden.

2.3.5 X-Content-Type-Options

X-Content-Type-Options: nosniff

Mit diesem Header wird dem Browser verboten, „MIME sniffing“⁵ 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 Ressourcen kontrollieren. Dadurch kann verhindert werden, dass JavaScript 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.

³Siehe <https://blog.innerht.ml/the-misunderstood-x-xss-protection/> für generelle Erklärung und Beispiel des Angriffs zum Entfernen der JQuery-Bibliothek

⁴<https://bugs.chromium.org/p/chromium/issues/detail?id=825675>

⁵Beim „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.

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, T24	✓
Ausgabecodierung	T6, T26, T31	✓
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 „Content Security Policy“-Header nutzen	T6, T31	✓
„X-XSS-Protection“-Header nutzen	T6	✓
<u>Speicherplatzlimit auf Dateinamen und Kommentarfeld ausweiten</u>	T21	✓
<u>Hervorhebung der Nutzer-eigenen Datei</u>	T22	✓
<u>Beim Download einer Datei HTTP-Header Content-Type auf application/octet-stream setzen</u>	T23	✓
<u>Maximal-/Minimalwert für Session-Timeout definieren</u>	T32, T33, T34	✓
<u>Return-URL via Regex überprüfen</u>	T26	✓
<u>Funktionalität, die Return-URL benötigt, via HTTP POST abfeuern</u>	T26	✓

Abbildungsverzeichnis

1.1 Gesamtübersicht des Datenflusses in unserer Applikation	3
---	---