FILE HAVEN

Sistema sicuro di condivisione file .txt

Corso di Laurea Magistrale in Sicurezza Informatica – Sede di Taranto Insegnamento: Sicurezza nelle Applicazioni
Anno Accademico: 2025-2026
Docenti:

- Prof. Ardimento Pasquale
- Prof. Malerba Donato

Team “Itopacci”

Cognome & Nome	Matricola	Email
Mongelli Antonio	856645	a.mongelli47@studenti.uniba.it
Pagliaro Francesco	856646	f.pagliaro@studenti.uniba.it

1. Introduzione

1.1 Obiettivo del Progetto

L'obiettivo del progetto è lo sviluppo di **"File Haven"**, un'applicazione web Java EE per la gestione sicura di documenti testuali. Il sistema è progettato per mitigare le vulnerabilità critiche definite dall'OWASP Top 10, garantendo riservatezza, integrità e disponibilità, in conformità con i requisiti **RF1-RF7** e i requisiti di sicurezza specifici (**3.1-3.9**).

1.2 Stack Tecnologico

- **Linguaggio:** Java 8 (Servlet API 4.0.1, JSTL 1.2).
- **Database:** H2 Engine (Embedded) file-based.
- **Librerie di Sicurezza:**
  - jBCrypt (0.4): Hashing delle password con salt automatico e work factor adattivo.
  - Apache Tika (2.6.0): Validazione del contenuto reale dei file (magic numbers).
  - OWASP Java Encoder: Sanitizzazione degli output per prevenzione XSS.
- **Container:** Apache Tomcat 9 / Jetty.

2. Analisi Statica (Architettura e Sicurezza)

In questa sezione vengono descritte le scelte progettuali adottate per garantire la sicurezza *by-design*.

2.1 Gestione delle Credenziali e Password Policy (RF1)

Le password non sono mai salvate in chiaro. Sono gestite tramite l'algoritmo BCrypt (cost

factor 12), che protegge da attacchi Rainbow Table e Brute Force.
La validazione impone una politica di complessità tramite Regex:
^(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%^&+=!])(?=\\S+$).{8,}$

2.2 Gestione Sessioni e Cookie (RF2, RF4, RF7)

L'applicazione implementa una configurazione sicura dei cookie centralizzata nel web.xml e rafforzata da un filtro programmatico (SameSiteCookieFilter) per garantire compatibilità tra container diversi.

- **HttpOnly:** Impedisce l'accesso ai cookie via JavaScript (mitigazione XSS).
- **SameSite=Strict:** Impedisce l'invio di cookie in richieste cross-site (mitigazione CSRF).
- **Secure:** Abilitato (impostabile a true per ambienti di produzione HTTPS).
- **Rigenerazione ID (Session Fixation):** Al login (LoginServlet) la vecchia sessione viene invalidata e ne viene creata una nuova.

Configurazione web.xml:

```
<cookie-config>
   <http-only>true</http-only>
   <secure>false</secure> <!-- True in produzione con HTTPS -->
   <same-site>Strict</same-site>
</cookie-config>
<session-config>
   <session-timeout>30</session-timeout>
</session-config>
```

2.3 Upload Sicuro e Validazione Contenuto (RF5)

Il sistema implementa una strategia "Defense in Depth" per l'upload:

1. **Whitelist Estensioni:** Solo .txt accettati.
2. **Analisi Tika:** Ispezione dei "Magic Number" per bloccare eseguibili mascherati (es. .exe rinominati in .txt).
3. **Storage Isolato:** I file sono salvati nella directory ~/secure-app-uploads/, esterna alla web-root, prevenendo l'esecuzione diretta via URL (RCE).

2.4 Protezione da Injection e XSS (RF6)

- **SQL Injection:** Tutte le interazioni col database (UserDAO, FileDAO) utilizzano esclusivamente PreparedStatement, separando la logica dai dati.
- **XSS (Cross-Site Scripting):** L'output utente (nomi file, contenuto) è codificato usando OWASP Java Encoder prima del rendering. Inoltre, l'header Content-Type: text/plain e X-Content-Type-Options: nosniff vengono forzati durante la visualizzazione dei file.

2.5 Gestione della Concorrenza (RF 3.8)

Utilizzo di ConcurrentUploadService.java per prevenire vulnerabilità TOCTOU (*Time-of-Check to Time-of-Use*) e Race Conditions:

- **ReentrantLock:** Garantisce l'atomicità dell'operazione di verifica e scrittura del file.
- **ExecutorService:** Pool di thread gestito per prevenire la Resource Exhaustion durante upload multipli.
- **AtomicLong:** Generazione thread-safe di nomi file univoci.

3. Guida al Deployment

3.1 Procedura di Build

L'applicazione è gestita tramite Maven. Per generare l'artefatto:
$env:JAVA_HOME = "C:\Percorso\JDK"
.\tools\maven\bin\mvn.cmd clean package

L'artefatto generato è target/secure-web-app.war.

4. Analisi Dinamica e Piano di Testing

Di seguito i risultati dei test effettuati secondo il piano di test (rif. TESTING.md).

4.1 Test d'Uso (Funzionalità Corrette)

ID	Test	Descrizione	Esito
TU1	Registrazione	Creazione account con hashing BCrypt e policy robusta	✅ PASS
TU2	Login	Accesso riuscito e rigenerazione Session ID	✅ PASS
TU3	Login Errato	Accesso negato con credenziali invalide (messaggio generico)	✅ PASS
TU4	Area Riservata	Accesso consentito alla dashboard con sessione valida	✅ PASS
TU5	Upload TXT	Caricamento riuscito di un file di testo valido	✅ PASS
TU6	Visualizzazione	Contenuto mostrato in sicurezza (text/plain), script non eseguiti	✅ PASS
TU7	Timeout	Redirect a login dopo 30 min inattività	✅ PASS
TU8	Logout	Sessione invalidata server-side, cookie rimosso	✅ PASS

TU9	Post-Logout	Accesso negato alle aree protette dopo il logout	✅ PASS
TU10	Concorrenza	Gestione sicura di upload simultanei (no sovrascritture)	✅ PASS

Nota per la documentazione:
Inserire qui gli screenshot relativi ai Test d'Uso.

- Screenshot consigliato: **TU2** (mostrare i cookie in DevTools)
- Screenshot consigliato: **TU10** (mostrare database con file caricati simultaneamente)

4.2 Test di Abuso (Security Testing)

ID	Attacco	Difesa Implementata	Esito
TA1	SQL Injection	Uso sistematico di PreparedStatement nei DAO	✅ BLOCCATO
TA2	Bypass Auth	AuthFilter intercetta richieste dirette senza sessione	✅ BLOCCATO
TA3	Estensione Vietata	Upload di file .exe o .pdf rifiutato	✅ BLOCCATO
TA4	Fake .txt	Apache Tika rileva header binario/exe in file rinominati	✅ BLOCCATO
TA5	Stored XSS	OWASP Java Encoder in rendering e Content-Type: text/plain	✅ MITIGATO
TA6	Forced Browsing	Tentativo di accesso anonimo a /dashboard bloccato	✅ BLOCCATO
TA7	Replay Sessione	Riutilizzo di cookie di sessione scaduta/invalidata fallisce	✅ BLOCCATO
TA8	Esecuzione File	Accesso diretto via URL alla cartella upload (404 Not Found)	✅ BLOCCATO

Nota per la documentazione:
Inserire qui gli screenshot relativi ai Test di Abuso.

- Screenshot consigliato: **TA1** (tentativo login con caratteri SQL)

- Screenshot consigliato: **TA4** (messaggio errore Tika)
- Screenshot consigliato: **TA5** (codice HTML visibile ma non eseguito)

5. Conclusioni

"File Haven" soddisfa pienamente i requisiti **RF1-RF7** e i requisiti di sicurezza aggiuntivi. L'architettura adotta un approccio *"Defense in Depth"*, combinando validazione rigorosa degli input, gestione sicura della sessione e protezione a livello di infrastruttura (storage isolato), garantendo la protezione dei dati utente contro le più comuni minacce web.

Appendice: Struttura del Progetto

```
src/main/java/com/secureapp/
├── dao/ (UserDAO, FileDAO - PreparedStatement & BCrypt)
├── filter/ (AuthFilter, SameSiteCookieFilter, SecurityHeadersFilter)
├── model/ (User, FileModel)
├── service/ (ConcurrentUploadService - Thread Safe)
├── servlet/ (LoginServlet, UploadServlet, FileContentServlet)
└── util/ (ValidationUtil, DatabaseUtil)
```