Progetto di Sicurezza nelle Applicazioni

1. Obiettivo

L'obiettivo del progetto è progettare e implementare un'applicazione web in Java seguendo i principi della **secure software development**, dimostrando la capacità di **identificare, comprendere e mitigare le vulnerabilità delle applicazioni web** studiate nel corso.

L'applicazione dovrà essere sviluppata applicando sistematicamente le **best practice di sicurezza**, con particolare attenzione alla gestione di:

- dati sensibili,
- autenticazione e sessioni utente,
- cookie,
- input non affidabili,
- caricamento di file.

Il progetto deve dimostrare non solo il corretto funzionamento delle funzionalità applicative, ma anche la capacità di **prevenire attacchi comuni** (quali SQL Injection, XSS, CSRF, session hijacking e upload di file malevoli) attraverso scelte progettuali consapevoli e implementazioni sicure.

2. Requisiti Funzionali

RF1 – Creazione account utente con credenziali robuste

L'applicazione deve consentire la creazione di un account utente mediante:

- indirizzo e-mail univoco e formalmente valido;
- password conforme a una policy di sicurezza (lunghezza minima, complessità, assenza di pattern banali).

Il sistema deve prevenire la creazione di account duplicati e fornire messaggi di errore non informativi per l'attaccante.

RF2 – Autenticazione sicura e avvio di sessione

L'applicazione deve consentire agli utenti registrati di autenticarsi tramite e-mail e password. A seguito del login:

- viene creata una nuova sessione HTTP;
- viene generato un identificatore di sessione sicuro;
- il sistema evita il riutilizzo di eventuali sessioni precedenti.

RF3 – Gestione della sessione utente

L'applicazione deve gestire la sessione dell'utente autenticato garantendo che:

- le risorse riservate siano accessibili solo con sessione valida;

- la sessione abbia un time-out configurabile;
- l'accesso venga negato in caso di sessione scaduta o assente.

RF4 – Gestione dei cookie di sessione

L'applicazione deve utilizzare cookie per la gestione della sessione impostando correttamente gli attributi di sicurezza:

- `HttpOnly`
- `Secure`
- `SameSite`

L'utente non deve poter leggere o modificare direttamente i cookie lato client.

RF5 – Caricamento controllato di contenuti da parte dell'utente

Un utente autenticato può caricare contenuti testuali associati al proprio account, nel rispetto delle seguenti condizioni:

- sono ammessi esclusivamente file di tipo testuale (`.txt`);
- i contenuti caricati vengono validati prima dell'elaborazione;
- il sistema impedisce il caricamento di file non conformi.

RF6 – Visualizzazione sicura dei contenuti condivisi

Gli utenti autenticati possono visualizzare i contenuti caricati da altri utenti. La visualizzazione deve avvenire in modo sicuro, garantendo che:

- il contenuto non venga interpretato come codice eseguibile;
- eventuali caratteri speciali o markup vengano correttamente gestiti.

RF7 – Terminazione sicura della sessione (logout)

L'applicazione deve consentire all'utente di terminare esplicitamente la sessione. Il logout deve comportare:

- invalidazione della sessione server-side;
- eliminazione o invalidazione dei cookie di sessione;
- impossibilità di riutilizzare la sessione precedente.

RF8 – Gestione dei messaggi di feedback all'utente

L'applicazione deve fornire messaggi di feedback chiari e coerenti per ogni operazione rilevante, senza rivelare informazioni sensibili sul funzionamento interno del sistema.

Esempi:

- esito della registrazione;
- esito del login;
- esito del caricamento di contenuti;
- stato della sessione (scaduta, terminata, non valida).

3. Requisiti di sicurezza da soddisfare

L'applicazione deve integrare in modo sistematico le seguenti misure di sicurezza. Ogni requisito deve essere **effettivamente implementato**, **verificabile a run-time** e **documentato** nella relazione finale.

3.1 Gestione sicura dei cookie

L'applicazione deve utilizzare i cookie in modo sicuro, garantendo che non diventino un vettore di attacco. In particolare, deve essere implementato quanto segue:

- **Impostazione dei cookie**
  I cookie devono essere creati impostando correttamente gli attributi di sicurezza:
  - `HttpOnly`
  - `Secure`
  - `SameSite` (con scelta motivata del valore)
- **Ricezione e utilizzo dei cookie**
  I cookie devono essere utilizzati esclusivamente per le finalità previste (es. gestione della sessione), evitando esposizioni che possano facilitare attacchi XSS o session hijacking.
- **Protezione dei dati contenuti nei cookie**
  Eventuali informazioni sensibili non devono essere memorizzate in chiaro; se necessario, devono essere protette tramite tecniche di cifratura o firma.
- **Gestione del ciclo di vita dei cookie**
  Devono essere correttamente gestiti:
  - scadenza (`Max-Age` / `Expires`);
  - invalidazione dei cookie non più validi;
  - cancellazione in fase di logout.

3.2 Gestione sicura della sessione HTTP

L'applicazione deve implementare una gestione robusta delle sessioni utente. In particolare:

- la sessione deve essere creata solo dopo autenticazione riuscita;
- deve essere impostato un **timeout di sessione appropriato**;
- l'identificatore di sessione deve essere rigenerato nei momenti critici (es. login);
- devono essere prevenuti attacchi di:
  - session fixation;
  - session hijacking;
  - riutilizzo di sessioni scadute o invalidate.

3.3 Caricamento sicuro dei file

Il caricamento di file da parte degli utenti deve essere gestito in modo difensivo. In particolare:

- i file devono essere validati **in base al contenuto reale**, non solo all'estensione;
- deve essere utilizzata una libreria di analisi del contenuto (es. **Apache Tika**);
- devono essere accettati solo i formati esplicitamente consentiti;
- i file devono essere salvati in directory **non eseguibili** e **non direttamente accessibili dal web**;
- deve essere gestita correttamente la vulnerabilità **TOCTOU (Time-of-Check to Time-of-Use)**, separando le fasi di controllo e utilizzo.

3.4 Protezione contro injection e input malevoli

L'applicazione deve prevenire attacchi di injection e l'uso di input non affidabili. In particolare:

- tutte le interazioni con il database devono utilizzare **Prepared Statements**;
- nessuna query deve essere costruita tramite concatenazione di stringhe;
- i dati in ingresso devono essere validati tramite:
  - whitelisting quando possibile;
  - blacklisting solo come supporto;
  - escaping e output encoding per la visualizzazione dei dati.

3.5 Gestione sicura delle chiavi e crittografia

L'applicazione deve proteggere adeguatamente le chiavi crittografiche e i dati sensibili. In particolare:

- devono essere utilizzati algoritmi di cifratura sicuri (es. **AES**);
- le chiavi non devono essere hardcoded nel codice sorgente;
- le chiavi devono essere gestite tramite meccanismi sicuri (es. **keystore**);
- la cifratura deve essere utilizzata solo dove necessario e in modo corretto.

3.6 Gestione sicura delle credenziali utente

Le password e le credenziali devono essere trattate come dati altamente sensibili. In particolare:

- le password devono essere memorizzate esclusivamente sotto forma di hash;
- deve essere utilizzato un algoritmo di hashing robusto con salt (es. **PBKDF2**, **bcrypt**);
- le password non devono mai essere memorizzate o trasmesse in chiaro;
- il sistema deve prevenire tecniche di enumerazione degli utenti.

3.7 Programmazione difensiva e qualità del codice

L'applicazione deve essere sviluppata seguendo i principi della **programmazione difensiva**. In particolare:

- riduzione dello scope delle variabili al minimo necessario;
- uso corretto dei modificatori di accesso (`private`, `protected`, `default`);
- applicazione del principio di **information hiding**;
- assenza di codice ridondante o potenzialmente pericoloso;
- gestione esplicita degli errori senza esporre informazioni sensibili.

3.8 Concorrenza e consistenza delle risorse di file

L'applicazione Web da realizzare deve includere almeno una funzionalità in cui l'accesso concorrente alle risorse di file venga gestito esplicitamente a livello applicativo mediante meccanismi di concorrenza del linguaggio Java. In particolare, l'applicazione deve prevedere l'esecuzione concorrente di operazioni di gestione dei file (ad esempio durante il caricamento o l'elaborazione dei contenuti) attraverso la creazione esplicita di thread o l'utilizzo di API concorrenti fornite dalla piattaforma Java. **L'accesso a risorse condivise quali**:

- nomi dei file generati lato server
- directory di destinazione
- file temporanei o risorse intermedie

deve essere protetto mediante opportuni meccanismi di sincronizzazione (ad esempio synchronized, lock espliciti o strutture concorrenti), al fine di prevenire condizioni di race, sovrascritture non intenzionali e stati intermedi inconsistenti. L'uso dei meccanismi di concorrenza deve essere motivato, corretto e limitato alle sole sezioni critiche del codice, in coerenza con i principi di programmazione difensiva e sviluppo sicuro.

4. Criteri di valutazione

La valutazione del progetto terrà conto sia della **correttezza funzionale** dell'applicazione sia della **qualità delle soluzioni di sicurezza adottate**, con particolare attenzione alla capacità dello studente di applicare in modo consapevole i concetti teorici illustrati nel corso.

4.1 Correttezza dell'implementazione delle misure di sicurezza

Verrà valutata la corretta applicazione delle tecniche di sicurezza richieste, tra cui:

- gestione sicura dei cookie (impostazione degli attributi `HttpOnly`, `Secure`, `SameSite`);
- protezione contro attacchi di injection (uso di Prepared Statements);
- gestione corretta delle sessioni HTTP (timeout, rigenerazione dell'identificatore, prevenzione della session fixation);
- caricamento sicuro dei file e prevenzione di upload malevoli;
- prevenzione di vulnerabilità XSS e CSRF.

4.2 Qualità del codice e programmazione difensiva

Verranno valutati:

- chiarezza, leggibilità e organizzazione del codice sorgente;
- uso appropriato dei principi di **programmazione difensiva**;
- corretta applicazione dei principi di **information hiding**;
- riduzione dello scope delle variabili e uso coerente dei modificatori di accesso;
- assenza di codice ridondante o pericoloso dal punto di vista della sicurezza.

4.3 Approccio progettuale alla sicurezza

Verrà valutata la capacità di:

- individuare i punti critici dell'applicazione dal punto di vista della sicurezza;
- adottare contromisure adeguate e proporzionate al contesto;
- progettare l'applicazione considerando la sicurezza come requisito funzionale, e non come aggiunta successiva;
- prevenire attacchi comuni attraverso scelte architetturali consapevoli.

4.4 Documentazione e giustificazione delle scelte

La documentazione sarà valutata in base a:

- chiarezza e completezza della descrizione delle soluzioni adottate;
- capacità di collegare le scelte implementative ai principi teorici studiati;
- corretta descrizione dei test d'uso e di abuso effettuati;
- coerenza tra codice, comportamento dell'applicazione e documentazione fornita.

4.5 Coerenza complessiva e aderenza alla traccia

Verrà infine valutata:

- la coerenza complessiva del progetto;
- l'aderenza ai requisiti funzionali e di sicurezza definiti nella traccia;
- la capacità di rispettare le linee guida fornite e di produrre un lavoro completo e strutturato.

5. Consegna

L'applicazione deve essere consegnata sotto forma di **progetto Java completo**, comprensivo di codice sorgente, configurazioni e documentazione tecnica, che dimostri in modo esplicito come sono state **identificate, mitigate e verificate** le vulnerabilità di sicurezza studiate nel corso.

Artefatti software

- **Archivio** `.war` dell'applicazione web:
  - codice sorgente completo;
  - configurazioni (sessioni, cookie, upload, sicurezza);
  - commenti significativi sul codice di sicurezza.

Documentazione tecnica (`.docx`)

La documentazione deve essere strutturata nelle seguenti sezioni.

5.1 Analisi Statica (obbligatoria)

Descrizione delle **scelte progettuali e implementative** adottate per la sicurezza, con riferimento esplicito a:

- gestione dei cookie (`HttpOnly`, `Secure`, `SameSite`);
- gestione delle sessioni HTTP e prevenzione di session fixation;
- protezione da SQL Injection (Prepared Statements);
- validazione e sanitizzazione degli input;
- caricamento sicuro dei file (Apache Tika, TOCTOU);
- prevenzione di XSS (escaping/output encoding);
- gestione sicura delle password (salt + hashing);
- programmazione difensiva (scope delle variabili, access modifiers).
- gestione esplicita della concorrenza e sincronizzazione dell'accesso alle risorse di file (thread, lock, API concorrenti).

Ogni misura di sicurezza deve essere:

- descritta;
- motivata;
- collegata al codice che la implementa.

5.2 Analisi Dinamica (obbligatoria)

Documentazione del comportamento dell'applicazione **a run-time**, attraverso **test d'uso** e **test di abuso**, con evidenza del risultato ottenuto.

5.2.1 Test d'uso (Test funzionali corretti)

Devono essere documentati **almeno** i seguenti scenari di utilizzo corretto e, conseguentemente, i relativi scenari di utilizzo scorretto.

- TU1 – Creazione account con credenziali valide
  - Verifica che il sistema consenta la creazione di un nuovo account utente utilizzando un indirizzo e-mail valido e una password conforme alla policy di sicurezza definita, restituendo un messaggio di esito positivo.
- TU2 – Login con credenziali corrette
  - Verifica che un utente registrato possa autenticarsi correttamente fornendo credenziali valide e che venga avviata una nuova sessione HTTP.
- TU3 – Login con credenziali errate
  - Verifica che il sistema rifiuti un tentativo di autenticazione con credenziali non valide, senza fornire informazioni utili a un potenziale attaccante.
- TU4 – Accesso ad area riservata con sessione valida
  - Verifica che un utente autenticato e in possesso di una sessione valida possa accedere correttamente alle risorse riservate.
- TU5 – Caricamento contenuto testuale valido (.txt)
  - Verifica che un utente autenticato possa caricare correttamente un file di tipo testuale conforme ai requisiti richiesti (.txt) e che il contenuto venga accettato dal sistema.
- TU6 – Visualizzazione sicura dei contenuti caricati
  - Verifica che i contenuti caricati dagli utenti siano visualizzati correttamente senza essere interpretati come codice eseguibile e senza introdurre vulnerabilità di tipo XSS.
- TU7 – Scadenza della sessione (timeout)
  - Verifica che, allo scadere del tempo di inattività configurato, la sessione venga invalidata e l'accesso alle risorse riservate venga negato.
- TU8 – Logout corretto
  - Verifica che l'operazione di logout comporti l'invalidazione della sessione lato server e l'impossibilità di utilizzare la sessione precedentemente attiva.
- TU9 – Tentativo di accesso post-logout (negato)
  - Verifica che, dopo il logout, un tentativo di accesso a risorse riservate venga correttamente bloccato dal sistema.
- TU10 – Caricamento concorrente di contenuti testuali (gestione esplicita della concorrenza)
  - Il sistema deve gestire l'esecuzione concorrente di operazioni di gestione dei file attraverso la creazione esplicita di più thread o l'utilizzo di API concorrenti del linguaggio Java all'interno della stessa funzionalità di upload. Le operazioni concorrenti devono accedere alle medesime risorse di file (ad esempio nomi dei file, directory di destinazione o file temporanei), rendendo necessaria la sincronizzazione dell'accesso alle risorse condivise.
  - Comportamento atteso
    - le operazioni concorrenti non producono sovrascritture o interferenze tra file;
    - le risorse di file mantengono uno stato consistente;
    - i meccanismi di sincronizzazione adottati impediscono condizioni di race e stati intermedi inconsistenti.
  - Nota didattica: ai fini del presente test, la concorrenza deve essere introdotta e controllata a livello applicativo, mediante la creazione esplicita di thread o l'uso di API concorrenti Java, e non affidata esclusivamente alla simultaneità delle richieste HTTP generate dal client. La finalità del test è verificare la comprensione dei problemi di concorrenza e delle relative contromisure di sicurezza, nonché la corretta applicazione dei meccanismi di sincronizzazione sulle risorse condivise.

5.2.2 Test di abuso (attacchi simulati)

Devono essere documentati **almeno** i seguenti tentativi di attacco e il relativo comportamento difensivo dell'applicazione:

- **TA1** Tentativo di SQL Injection nel campo e-mail del login
- **TA2** Tentativo di bypass dell'autenticazione
- **TA3** Upload di file con estensione vietata (`.exe`, `.pdf`)
- **TA4** Upload di file con estensione lecita ma contenuto malevolo
- **TA5** Upload di contenuto testuale con script per stored XSS
- **TA6** Accesso a risorse protette senza sessione valida
- **TA7** Riutilizzo di cookie o sessione scaduta
- **TA8** Tentativo di esecuzione di file caricati

Per ciascun test devono essere indicati:

- input fornito;
- comportamento atteso;
- comportamento osservato;
- contromisura applicata.

5.3 Requisiti di qualità della consegna

- Il codice **non deve contenere implementazioni volutamente vulnerabili**.
- I test di abuso devono dimostrare che:
  - l'attacco è stato considerato;
  - l'applicazione reagisce in modo sicuro.
- La documentazione deve essere:
  - chiara;
  - coerente con il codice;
  - allineata ai contenuti del corso.

6. Riferimenti

Le scelte progettuali devono fare riferimento alle linee guida illustrate durante il corso e, ove opportuno, al testo: F. Long, D. Mohindra, R.C. Seacord, D. F. Sutherland, D. Svoboda. *Java Coding Guidelines*. Addison-Wesley, 2014.