



Secure Software Design: Design Pattern per la Sicurezza by Design

Michael Pio Stolfi 68787

Antonella Bonelli 68791

Introduzione

Obiettivo del progetto

- Dimostrare come l'uso dei design pattern possa migliorare la sicurezza di un'applicazione web

Metodologia

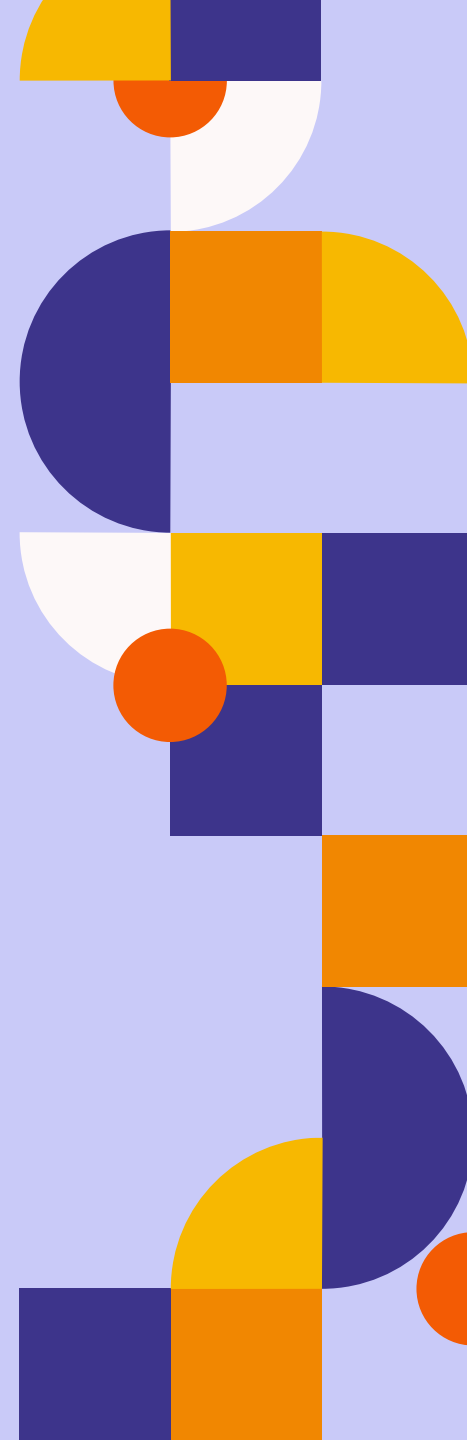
- Sviluppo di due backend Java con il framework Quarkus (uno insicuro e uno sicuro con design pattern)

Analisi

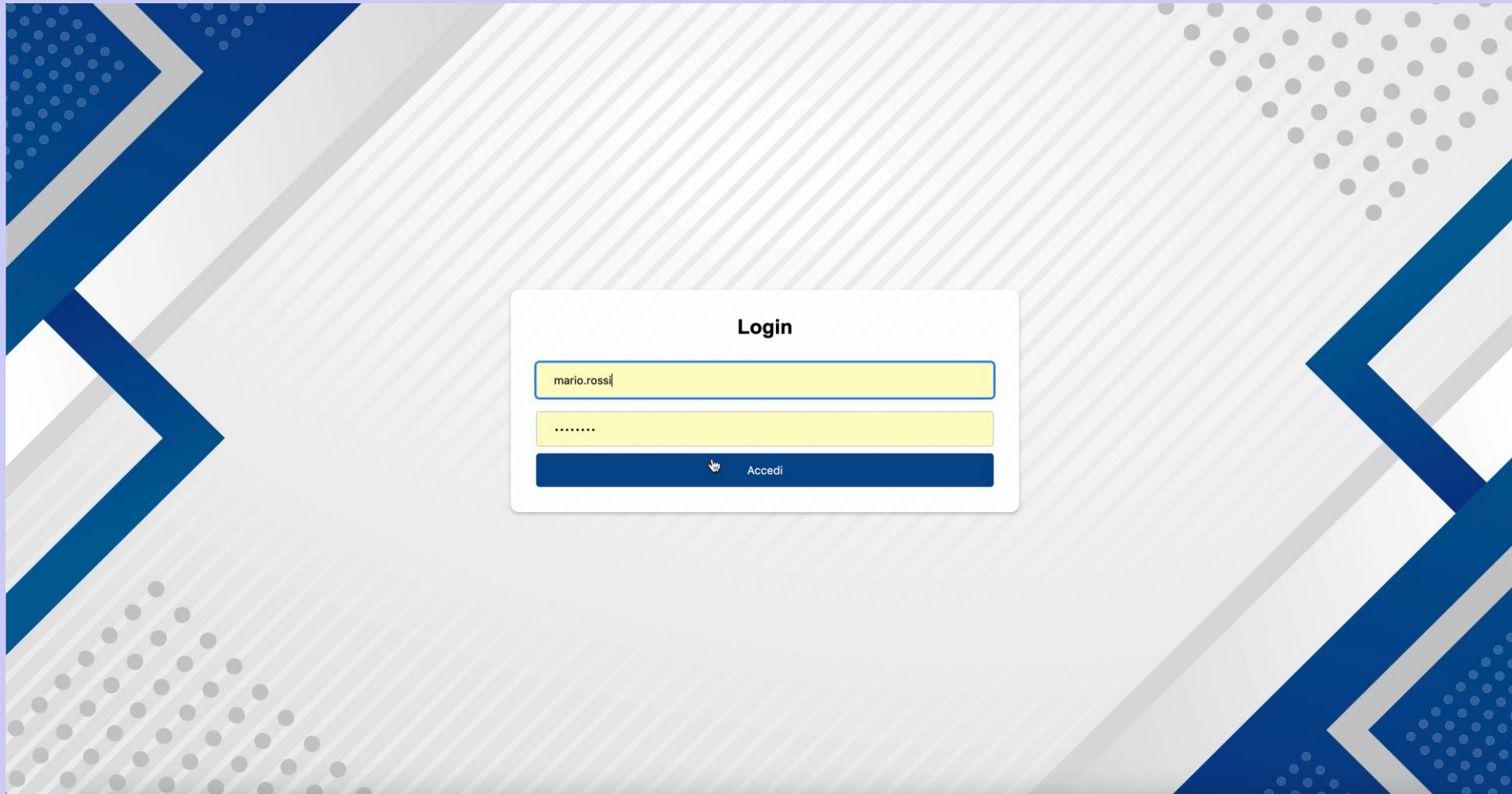
- Condotta attraverso test di sicurezza focalizzati sulle vulnerabilità più critiche identificate dall'OWASP Top 10

Risultati attesi

- Evidenziare il ruolo dei pattern nella mitigazione delle vulnerabilità e nell'implementazione della Security by Design




Introduzione

A login form is centered on a background with abstract geometric patterns in blue, grey, and white. The form is a white rectangle with a thin blue border. It contains a title, two input fields, and a submit button.

Login

mario.rossi

 Accedi

Casi d'uso sviluppati e rischi associati

Login

- Broken Access Control
- SQL Injection
- Identification and Authentication Failures
- Security Logging and Monitoring Failures

Visualizza dati utente nella Dashboard

- Security Logging and Monitoring Failures

Modifica dati utente nella Dashboard

- Broken Access Control
- SQL Injection
- Security Logging and Monitoring Failures

Design Pattern sistema Demo Sicuro

Proxy

- Vulnerabilità mitigata: Broken Access Control
- Motivazione:
 - Centralizza i controlli di accesso, evitando implementazioni distribuite e incoerenti
 - Previene l'accesso non autorizzato verificando il ruolo dell'utente prima di inoltrare le richieste
- Utilizzo: proteggere le operazioni di modifica dei dati



Design Pattern sistema Demo Sicuro

DAO

- Vulnerabilità ridotta: SQL Injection
- Motivazione:
 - Separazione della persistenza
 - Riduzione della vulnerabilità agli attacchi di SQL injection
 - Manutenibilità e scalabilità
- Utilizzo: gestione della persistenza delle utenze



Design Pattern sistema Demo Sicuro

Decorator

- Vulnerabilità ridotta: Identification and Authentication Failures
- Motivazione: è possibile aggiungere controlli di sicurezza senza modificarne l'implementazione centrale dell'autenticazione
- Utilizzo: gestire il blocco delle utenze a seguito di login falliti



Design Pattern sistema Demo Sicuro

Observer

- Vulnerabilità ridotta: Security Logging and Monitoring Failures
- Motivazione:
 - il sistema di monitoraggio centralizzato
 - migliora il disaccoppiamento e la scalabilità
- Utilizzo: sistema di monitoraggio degli accessi



Strategia di Testing

Obiettivo:

Verificare le differenze in termini di sicurezza tra il backend senza design pattern e quello con pattern di sicurezza

Approccio:

- Unit Test
- Penetration Test con Burp Suite

Strategia di Testing

	BE sicuro	BE insicuro
test_multiple_SQL_Injection_In_FindByUsername	✓	✗
test_sqlInjection_in_updateUser	✓	✗
test_SQL_Injection_In_FindByUsername	✓	✗
test_updateUser_with_null_or_empty_values	✓	✗

Strategia di Testing

Penetration Test

Test Eseguiti:

- **Test di Autenticazione** – Resistenza a brute force, gestione credenziali errate, blocco utenti
- **Test sulle Query SQL** – Verifica della protezione da SQL Injection
- **Test sui Permessi** – Analisi del controllo accessi per prevenire Broken Access Control
- **Test di Manipolazione Dati** – Tentativo di modifica non autorizzata dei livelli di accesso utente

Analisi dei Risultati

Backend senza Pattern

- Vulnerabile a brute force
- Suscettibile a SQL Injection
- Errori su valori nulli/lunghi
- Permessi aggirabili (Broken Access Control)

Backend con Pattern

- Blocco utenti dopo tentativi falliti
- Query parametrizzate impediscono attacchi di SQL Injection
- Controlli su input e validazione dati
- Verifica rigorosa dei ruoli utente

Linee Guida Applicazione dei Design Pattern

Proxy

Identificare operazioni e risorse critiche.
Centralizzare la validazione delle richieste.

Observer

Rilevare eventi di sicurezza e notificare osservatori (es. logging).
Garantire tracciabilità e facilità nell'aggiungere osservatori senza modificare il codice.

Decorator

Creare decorator di sicurezza (es. logout per login falliti).
Applicare i decorator in cascata senza modificare la logica di base.

DAO

Centralizzare la persistenza dei dati separando accesso e logica di business.
Usare query parametrizzate.
Validare gli input prima di passarli ai DAO.

Conclusioni

Risultati principali:

L'uso dei design pattern migliora la sicurezza dell'applicazione

Conclusione:

L'adozione di design pattern rappresenta una best practice nella sicurezza software, ma deve essere integrata in una strategia più ampia che combini principi di progettazione sicura e controlli di sicurezza diversificati