Università degli Studi di Bari Aldo Moro

Laurea Magistrale in Sicurezza Informatica

Sicurezza nelle Applicazioni

Caso di Studio A.A. 2025/26

Prof. Donato Malerba

Prof. Pasquale Ardimento

Dott. Matteo Esposito

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. Analisi Statica

2.1. Gestione dei Cookie

L'applicazione utilizza esclusivamente il cookie di sessione ***JSESSIONID*** per gestire lo stato dell'autenticazione e identificare l'utente tra le varie richieste HTTP.

I flag di sicurezza implementate sono:

- ***HttpOnly***: Il flag è impostato a true, questa misura impedisce agli script lato client di accedere al contenuto del cookie tramite ***document.cookie***. In caso di vulnerabilità XSS presente nell'applicazione, l'attaccante non potrà leggere il token di autenticazione, riducendo drasticamente il rischio di furto della sessione.
- ***Secure***: Il flag è impostato a true, garantisce che il cookie venga trasmesso dal browser al server esclusivamente su canali HTTPS. Questo previene attacchi di tipo Man-in-the-Middle o sniffing su reti non protette, dove il cookie potrebbe essere intercettato se inviato in chiaro su http.
- ***SameSite***: È stato scelto il valore ***Strict*** per garantire il livello più elevato di sicurezza. Il cookie viene inviato esclusivamente per richieste che hanno origine dallo stesso dominio. Questa configurazione elimina alla radice qualsiasi vettore di attacco Cross-Site Request Forgery.

Viene utilizzato ***Max-Age*** per definire la persistenza del cookie in secondi, preferito rispetto al deprecato ***Expires***.
L'applicazione sovrascrive il cookie esistente inviando un nuovo header con lo stesso nome e path, ma con ***Max-Age=0*** e valore vuoto, istruendo il browser a rimuoverlo immediatamente.

2.2. Gestione delle sessioni HTTP e prevenzione di session fixation

In ***LoginServlet.java***, sono state implementate specifiche misure difensive per mitigare i rischi di Session Fixation e Session Hijacking.
Il Session Fixation è un attacco in cui un malintenzionato impone un ID di sessione noto alla vittima prima che questa effettui il login. Se l'applicazione non rigenera l'ID al momento dell'autenticazione, l'attaccante può utilizzare quell'ID per accedere all'account della vittima.

Per prevenire questo scenario, la servlet applica la seguente logica al momento del login (metodo ***doPost***):

- Prima di autenticare l'utente, il sistema verifica l'esistenza di una sessione corrente tramite ***request.getSession(false)***.
- Se esiste una sessione, viene distrutta invocando ***oldSession.invalidate().*** Questo passaggio rimuove tutti i dati associati al vecchio ID sul server, rendendo inutile qualsiasi cookie di sessione che un attaccante potrebbe aver fissato nel browser della vittima.
- Viene creata una nuova sessione tramite ***request.getSession(true)***. Tomcat genera un nuovo JSESSIONID, che viene inviato al client. Questo garantisce che la sessione

autenticata abbia un identificativo completamente diverso da quello usato nella fase di pre-login.

Per ridurre la finestra di attacco in caso di Session Hijacking, è stato impostato un timeout di inattività server-side. Il comando ***newSession.setMaxInactiveInterval(30 * 60)***; impone che la sessione venga invalidata automaticamente dopo 30 minuti di inattività

2.3 Protezione da SQL Injection

La sicurezza dei dati è stata garantita implementando una difesa contro gli attacchi di tipo SQL Injection. Questa vulnerabilità si verifica quando l'input non fidato dell'utente viene concatenato dinamicamente alle stringhe delle query SQL, permettendo a un attaccante di alterare la logica del comando ed eseguire operazioni non autorizzate.
L'applicazione evita la costruzione dinamica delle query tramite concatenazione di stringhe. Tutte le interazioni con il database sono gestite esclusivamente tramite l'interfaccia ***java.sql.PreparedStatement***.

Le query SQL sono definite con dei segnaposto (?) al posto dei valori diretti. Questo indica al DBMS di trattare la struttura della query e i dati come entità separate fin dalla fase di precompilazione. I valori di input vengono associati ai segnaposto utilizzando i metodi setter tipizzati (***setString***, ***setBytes***, ***setInt***) forniti dall'oggetto ***PreparedStatement***. Grazie a questo meccanismo, il driver JDBC gestisce l'escape automatico dei caratteri speciali. Qualsiasi input fornito dall'utente, anche se contenente caratteri sintattici SQL, viene trattato dal database esclusivamente come dato letterale e mai come codice eseguibile.

2.4 Validazione e sanitizzazione degli input

L'applicazione adotta un approccio di Allow-listing, il sistema definisce i pattern di input accettabili. Tutto ciò che non corrisponde a questi pattern viene rifiutato a priori.
La classe ***ValidationUtil*** centralizza questa logica utilizzando le Espressioni Regolari (Regex) per:

- la verifica la conformità della e-mail allo standard RFC 5322.
- assicurare l'inserimento di una password complessa.

2.5 Caricamento sicuro dei file

La funzionalità di upload dei file rappresenta un punto critico per la sicurezza: se non adeguatamente protetta, potrebbe permettere a un attaccante di caricare file malevoli mascherati da file innocui, portando alla compromissione totale del server o ad attacchi lato client.
Per mitigare questi rischi, all'intero del controller ***UploadServlet*** è stata integrata la libreria ***Apache Tika***. Tika analizza il contenuto binario effettivo del file per determinarne il vero formato, ignorando completamente l'estensione e l'header inviato dal client.

2.6 Prevenzione di XSS

La sola validazione non è sufficiente a prevenire attacchi di tipo Cross-Site Scripting (XSS), in particolare la variante Stored XSS. Se un attaccante riuscisse a salvare un contenuto lecito ma

malevolo, la visualizzazione "grezza" di tale dato comprometterebbe gli utenti che visualizzano la pagina. L'applicazione evita l'uso di scriptlet Java diretti (<%= %>) nelle viste JSP, che scriverebbero l'output direttamente nel DOM senza controlli. È stata invece adottata la libreria JSTL (JavaServer Pages Standard Tag Library).

2.7 Gestione sicura delle password

Le password degli utenti non vengono mai memorizzate in chiaro nel database. L'architettura implementata adotta tecniche crittografiche per trasformare le password in stringhe irreversibili, rendendo le credenziali illeggibili anche in caso di compromissione totale del database. Per la gestione dell'hashing è stata selezionata la libreria ***BCrypt***, integrata nella classe ***PasswordUtil***.

La password viene trasformata in una stringa a lunghezza fissa tramite una funzione unidirezionale, questo permette al sistema di verificare la correttezza della password inserita al login confrontando gli hash, senza mai dover conoscere o salvare la password originale in chiaro. Un hash semplice è vulnerabile ad attacchi tramite dizionari Rainbow Tables. Il metodo ***BCrypt.gensalt(12)*** genera automaticamente un salt casuale e sicuro per ogni utente. Questo salt viene incluso nella stringa di hash finale salvata nel DB. Di conseguenza, due utenti con la stessa password avranno due hash completamente diversi nel database, rendendo inefficaci le Rainbow Tables.

Nel codice è stato configurato un Work Factor pari a 12. Questo parametro costringe la CPU a eseguire 2^12 iterazioni di hashing. Ciò rende la verifica di una singola password rapida per il server, ma rende computazionalmente proibitivo per un attaccante calcolare miliardi di tentativi al secondo per indovinare la password.

2.8 Programmazione difensiva

Oltre alle misure di sicurezza attive, l'applicazione è stata sviluppata seguendo i paradigmi della Programmazione Difensiva. Il codice è stato strutturato per ridurre al minimo lo scope delle variabili locali.

- Le variabili sono dichiarate il più vicino possibile al loro punto di utilizzo e il loro ciclo di vita è limitato al blocco di codice strettamente necessario.
- Ridurre lo scope impedisce il riutilizzo accidentale di variabili contenenti dati sensibili in parti di codice non autorizzate

L'applicazione applica il Principio del Privilegio Minimo anche alla struttura delle classi, utilizzando i modificatori di accesso per nascondere i dettagli implementativi interni.

- Tutte le variabili di istanza nelle classi Model sono dichiarate private. L'accesso a questi dati avviene esclusivamente tramite metodi getter/setter o costruttori, prevenendo la manipolazione diretta e inconsistente dello stato dell'oggetto da parte di codice esterno.
- Le classi di Util che non devono essere esposte all'API pubblica sono dichiarati private, impedendo che possano essere invocati impropriamente da altre parti dell'applicazione.

2.9 Gestione esplicita della concorrenza e sincronizzazione dell'accesso alle risorse di file

É stata implementata la classe di utilità ***FileManagerUtil*** questa componente gestisce operazioni su File System dove è critico prevenire conflitti di scrittura.

Due richieste simultanee potrebbero tentare di salvare un file con lo stesso nome nello stesso istante.
Senza sincronizzazione, si verificherebbe una vulnerabilità di tipo TOCTOU:

- Il Thread A controlla se il file esiste (file.exists()) -> Ritorna false.
- Il Thread B controlla se il file esiste -> Ritorna false.
- Il Thread A scrive il file.
- Il Thread B scrive il file, sovrascrivendo inconsapevolmente il lavoro del Thread A.

Per risolvere queste criticità, la classe ***FileManagerUtil*** implementa un meccanismo di Locking Esplicito. Questo garantisce un migliore incapsulamento della policy di sincronizzazione e previene che codice esterno possa interferire con il lock, causando potenziali deadlock.

Il blocco synchronized definisce la sezione critica. Solo un thread alla volta può eseguire la logica al suo interno. All'interno di questo blocco atomico, il sistema verifica l'esistenza del file e, in caso di conflitto, calcola un nome univoco incrementale (es. file_1.txt, file_2.txt) tramite un ciclo while. Grazie alla sincronizzazione, è matematicamente impossibile che due thread selezionino lo stesso nome "disponibile" contemporaneamente.

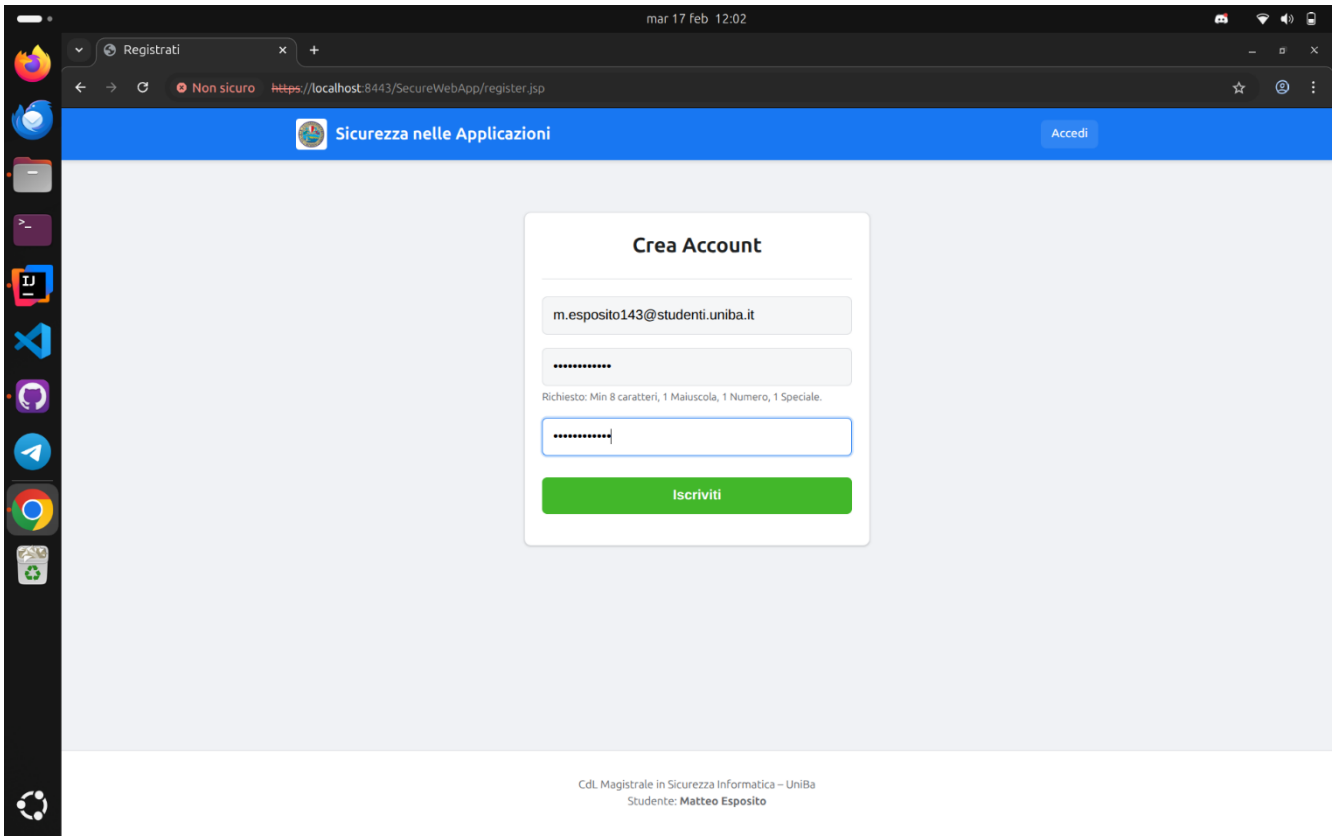
3. Analisi Dinamica

3.1 Test d'uso

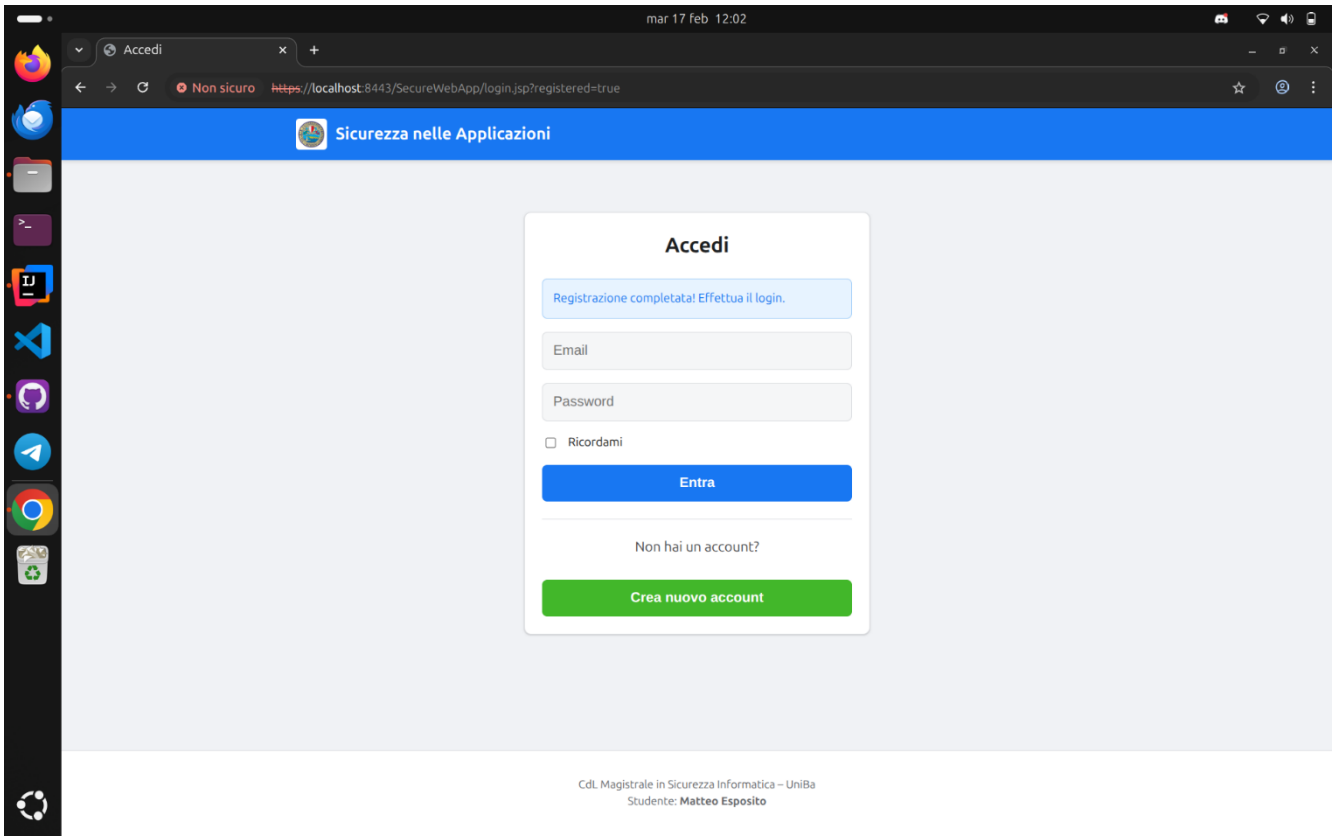
3.1.1 TU1 – Creazione account con credenziali valide

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.
Input fornito:

- e-mail: m.esposito143@studenti.uniba.it
- Password: Password123!
- Conferma Password: Password123!



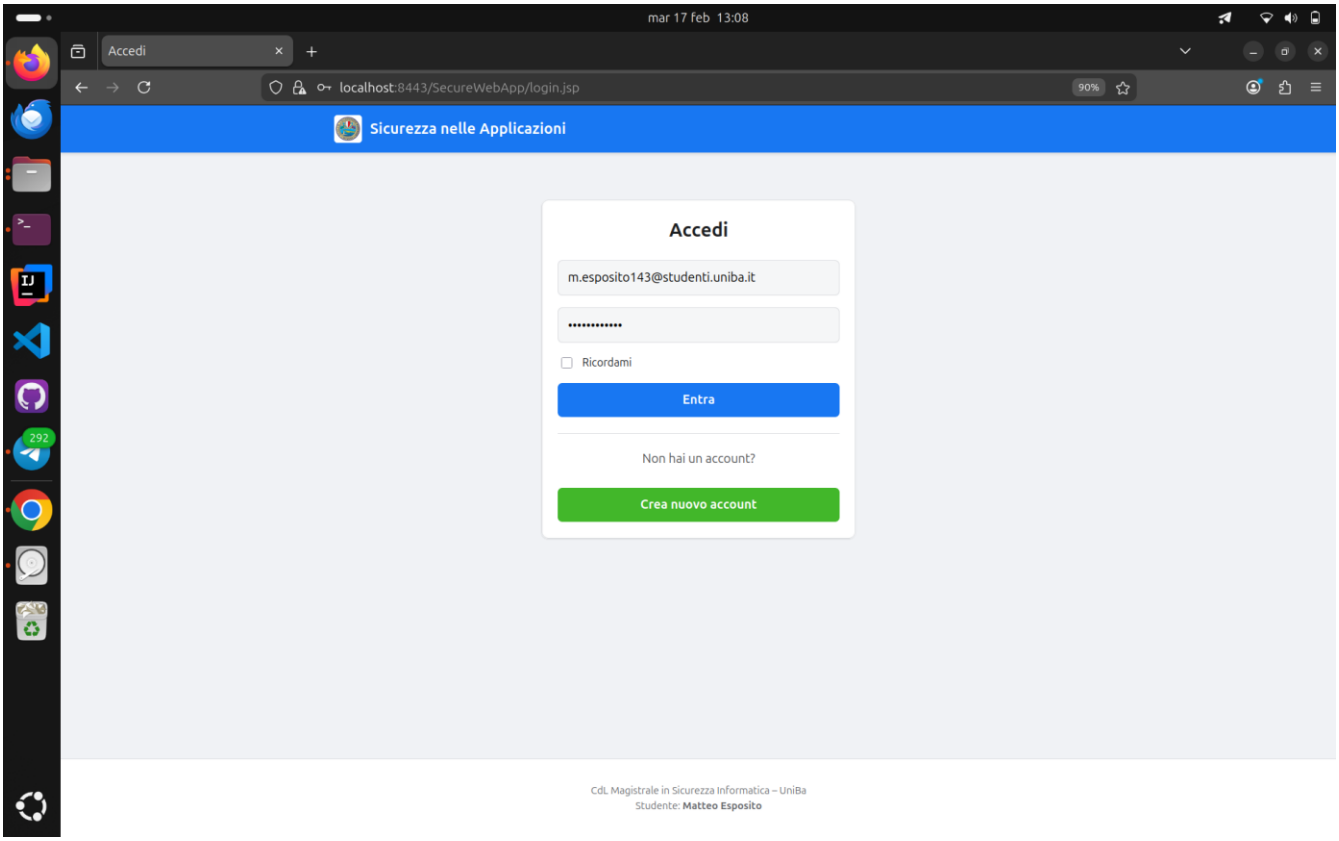
Comportamento atteso: Il sistema deve validare i dati, salvare l'utente nel database e reindirizzare alla pagina di login mostrando un messaggio di successo.
Comportamento osservato: L'applicazione ha accettato le credenziali e ha reindirizzato l'utente alla pagina ***login.jsp***, visualizzando correttamente il banner con il messaggio "Registrazione completata! Effettua il login.".


mar 17 feb 12:02
Accedi
Non sicuro
https://localhost:8443/SecureWebApp/login.jsp?registered=true
Sicurezza nelle Applicazioni
Accedi
Registrazione completata! Effettua il login.
Email
Password
Ricordami
Entra
Non hai un account?
Crea nuovo account
CdL Magistrale in Sicurezza Informatica – UniBa
Studente: Matteo Esposito

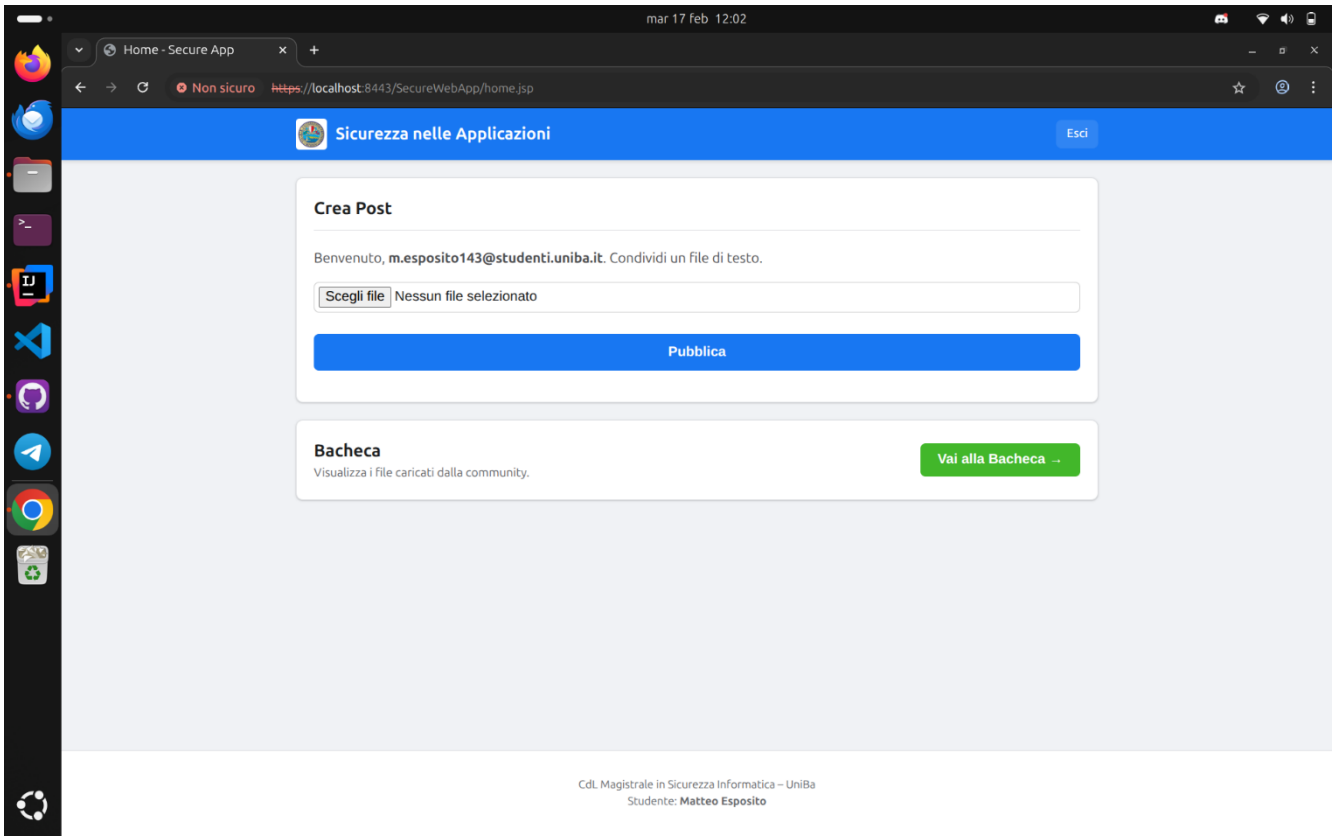
3.1.2 TU2 – Login con credenziali corrette

Verifica che un utente registrato possa autenticarsi correttamente fornendo credenziali valide e che venga avviata una nuova sessione HTTP.
Input fornito:

- e-mail: m.esposito143@studenti.uniba.it
- Password: Password123!



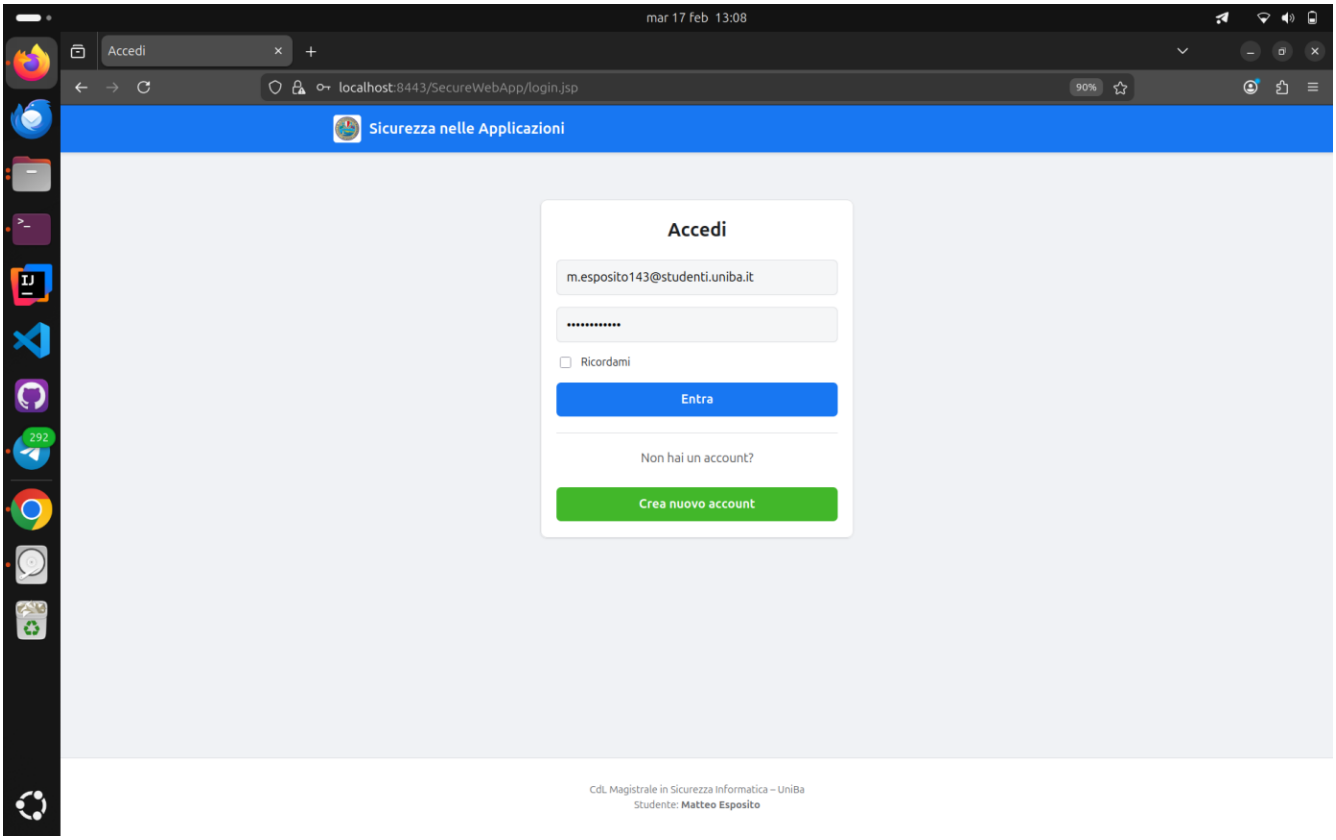
Comportamento atteso: Il sistema deve verificare la corrispondenza dell'hash della password. In caso di successo, deve invalidare la sessione anonima corrente e crearne una prima di reindirizzare l'utente all'area riservata (***home.jsp***).
Comportamento osservato: L'autenticazione è avvenuta con successo. Il sistema ha reindirizzato l'utente alla *Dashboard* personale mostrando il messaggio di benvenuto. Monitorando i cookie del browser, è stato riscontrato il cambio dell'identificativo di sessione.


mar 17 feb 12:02
Home - Secure App
Non sicuro
https://localhost:8443/SecureWebApp/home.jsp
Sicurezza nelle Applicazioni
Esci
Crea Post
Benvenuto, m.esposito143@studenti.uniba.it. Condividi un file di testo.
Scegli file
Nessun file selezionato
Pubblica
Bacheca
Visualizza i file caricati dalla community.
Vai alla Bacheca →
CdL Magistrale in Sicurezza Informatica – UniBa
Studente: Matteo Esposito

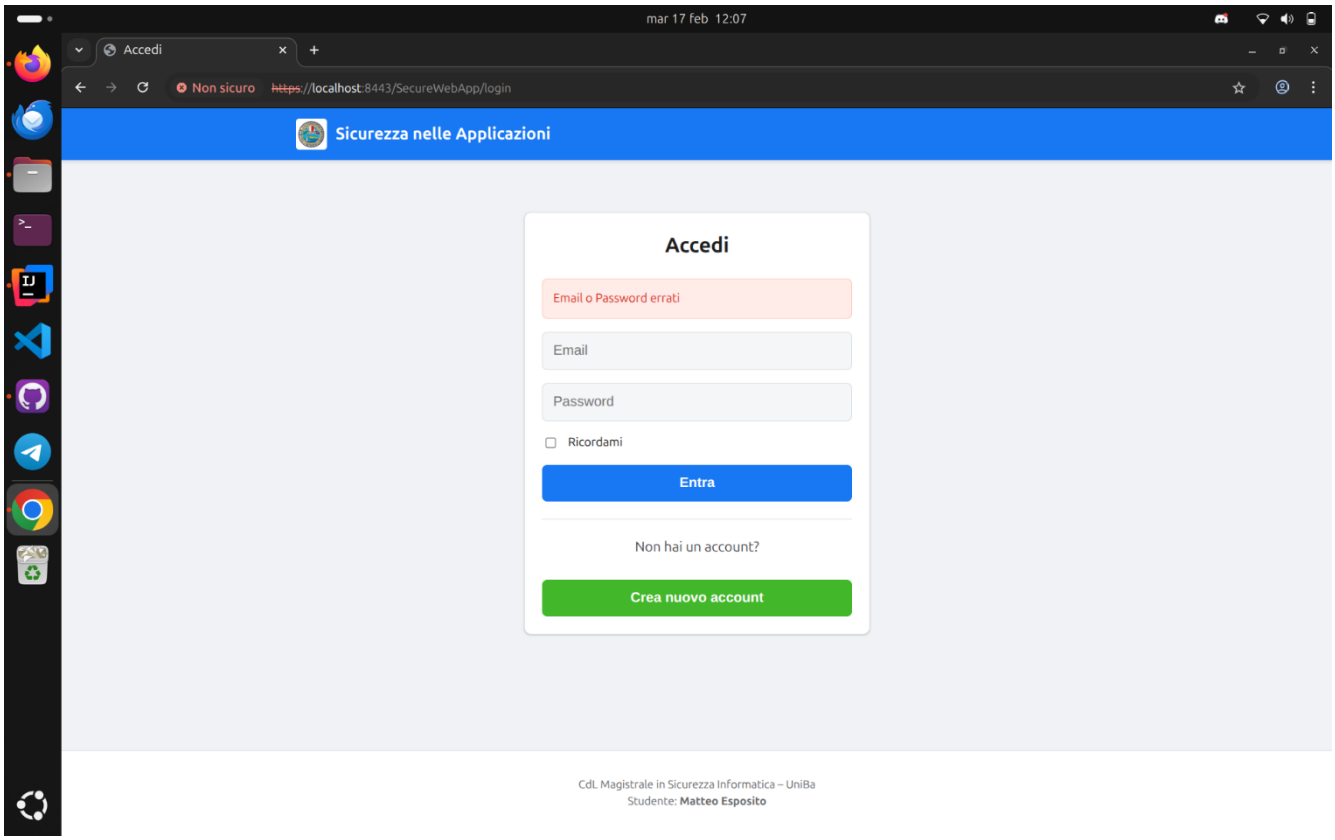
3.1.3 TU3 – Login con credenziali errate

Verifica che il sistema rifiuti un tentativo di autenticazione con credenziali non valide, senza fornire informazioni utili a un potenziale attaccante.
Input fornito:

- e-mail: m.esposito143@studenti.uniba.it
- Password: Password124!

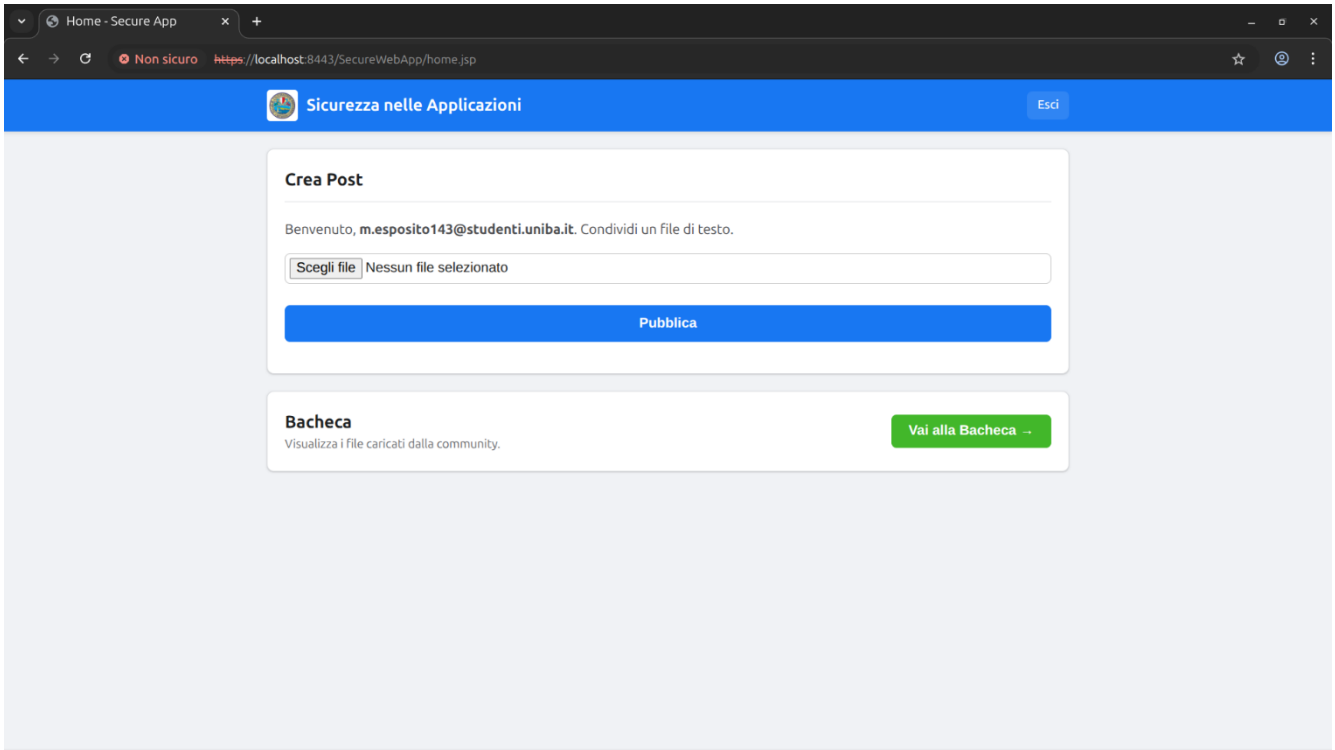


Comportamento atteso: Il sistema deve negare l'accesso rimanendo sulla pagina di login. Il messaggio di feedback mostrato all'utente deve essere ambiguo e non specifico, per non rivelare la presenza dell'account nel sistema.
Comportamento osservato: L'applicazione ha ricaricato la pagina ***login.jsp*** mostrando un banner di errore rosso con il testo generico "Email o Password errati".


mar 17 feb 12:07
Accedi
Non sicuro
https://localhost:8443/SecureWebApp/login
Sicurezza nelle Applicazioni
Accedi
Email o Password errati
Email
Password
Ricordami
Entra
Non hai un account?
Crea nuovo account
CdL Magistrale in Sicurezza Informatica – UniBa
Studente: Matteo Esposito

3.1.4 TU4 – Accesso ad area riservata con sessione valida

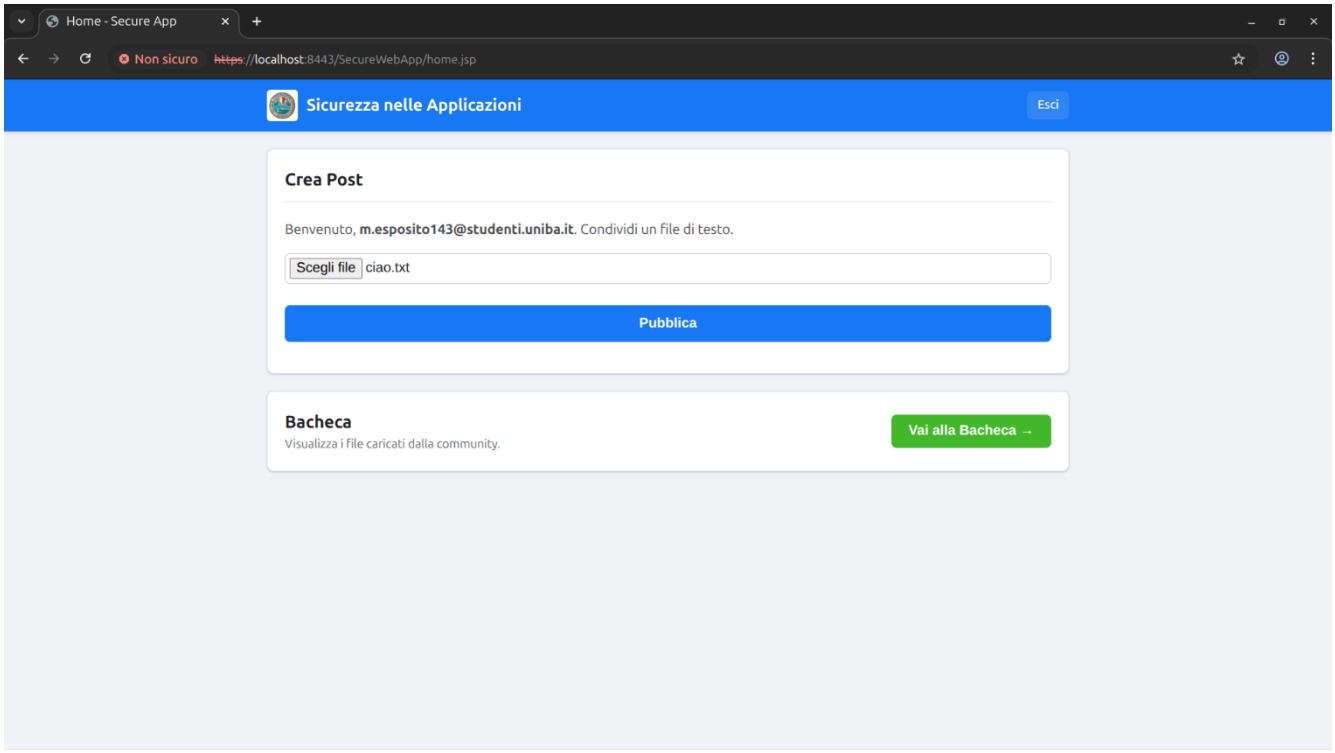
Verifica che un utente autenticato e in possesso di una sessione valida possa accedere correttamente alle risorse riservate.
Input fornito:
Richiesta HTTP GET verso una pagina protetta (es. https://localhost:8443/SecureWebApp/home.jsp).
Comportamento atteso: Il server deve intercettare la richiesta, validare l'identificativo di sessione e, confermando che corrisponde a un utente loggato, deve restituire la risorsa richiesta (codice 200 OK) mostrando i contenuti riservati.
Comportamento osservato: L'applicazione ha caricato correttamente la pagina ***home.jsp***.



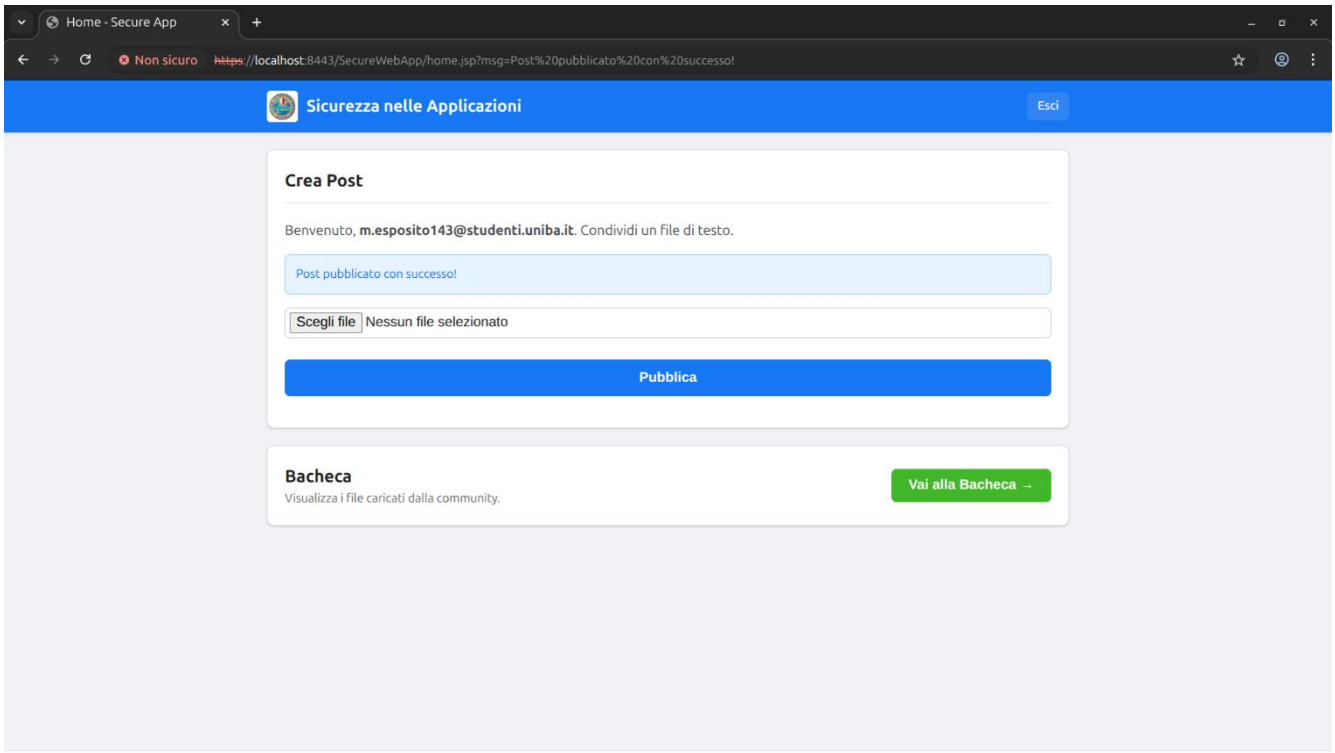
3.1.5 TU5 – Caricamento contenuto testuale valido

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.
Input fornito:

- File: ciao.txt.
- Contenuto: "Ciao".



Comportamento atteso: Il server deve analizzare i byte del file, riconoscere la firma text/plain, salvare la risorsa e restituire all'utente un messaggio di conferma.
Comportamento osservato: L'applicazione ha completato l'upload senza errori. È apparso il messaggio di successo e, verificando nella cartella di destinazione, il documento risulta presente e leggibile.

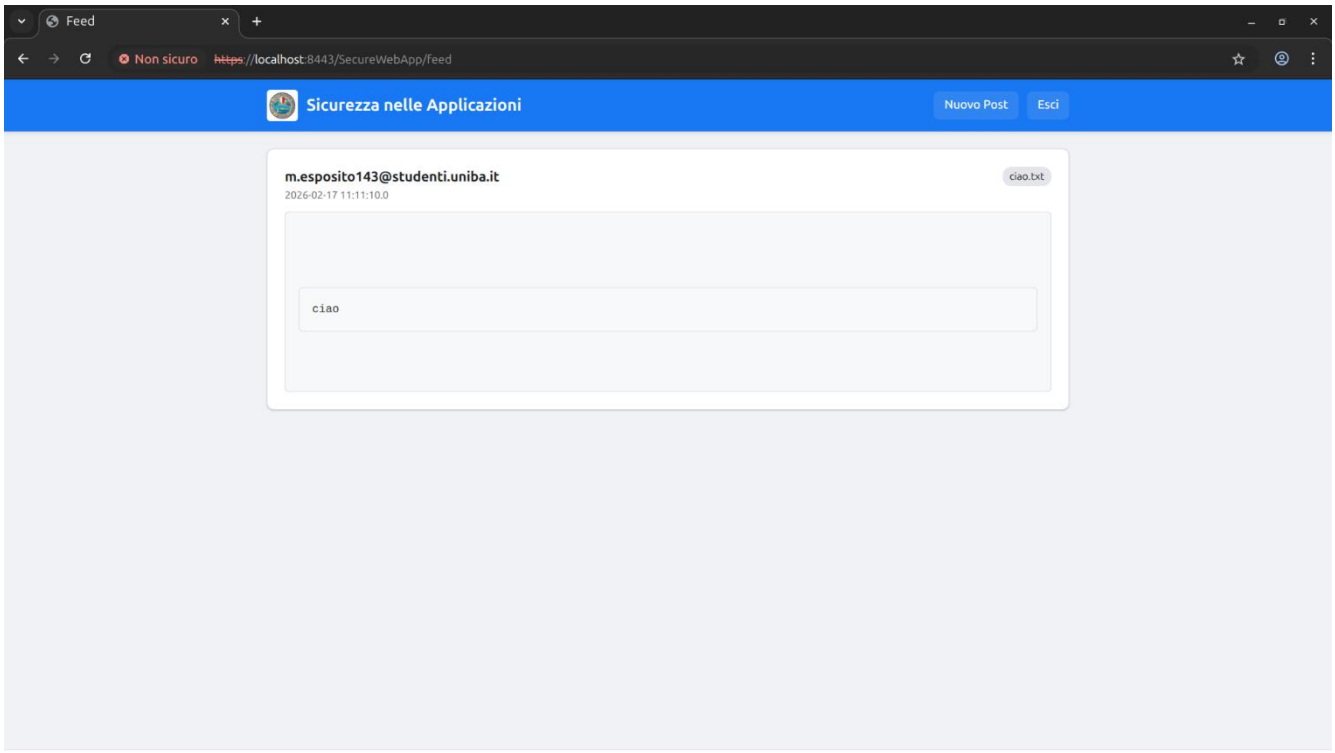

Home - Secure App
Non sicuro
https://localhost:8443/SecureWebApp/home.jsp?msg=Post%20pubblicato%20con%20successo!
Sicurezza nelle Applicazioni
Esci
Crea Post
Benvenuto, m.esposito143@studenti.uniba.it. Condividi un file di testo.
Post pubblicato con successo!
Scegli file
Nessun file selezionato
Pubblica
Bacheca
Visualizza i file caricati dalla community.
Vai alla Bacheca →

3.1.6 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.
Input fornito:

- Visualizzazione in bacheca del file ciao.txt caricato in TU5.

Comportamento atteso: Il browser deve visualizzare il file di testo.
Comportamento osservato: Accedendo alla pagina di visualizzazione (***feed.jsp***), viene riportato correttamente il testo contenuto nel file caricato in TU5.



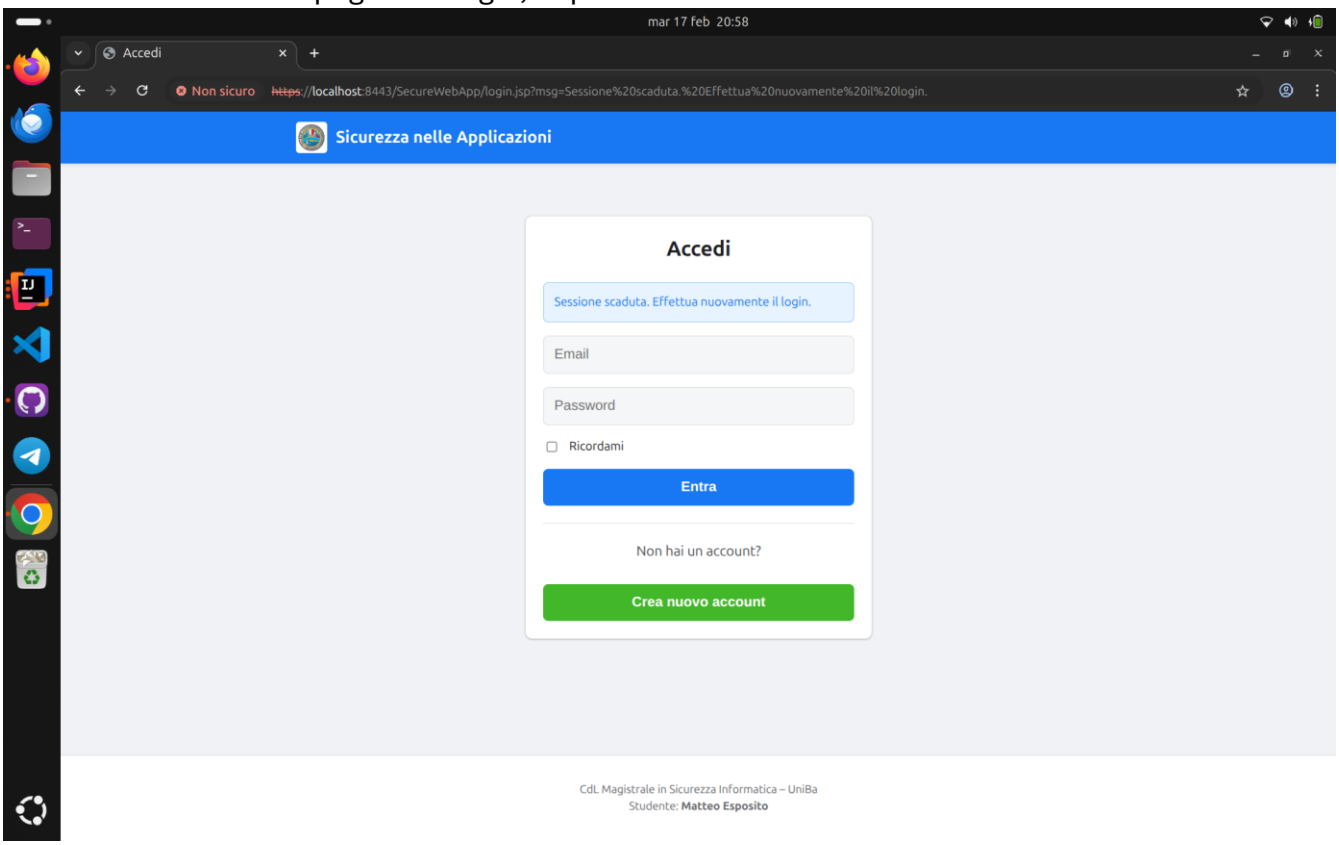
3.1.7 TU7 – Scadenza della sessione

Verifica che, allo scadere del tempo di inattività configurato, la sessione venga invalidata e l'accesso alle risorse riservate venga negato.

Input fornito:

- Esecuzione del login con credenziali valide.
- Attesa per un periodo superiore al session-timeout configurato.
- Tentativo di aggiornare la pagina o cliccare su un link riservato.

Comportamento atteso: Il server deve rilevare che il JSESSIONID inviato non è più associato a una sessione attiva. La richiesta deve essere intercettata e l'utente deve essere reindirizzato automaticamente alla pagina di Login, impedendo l'accesso alla risorsa.



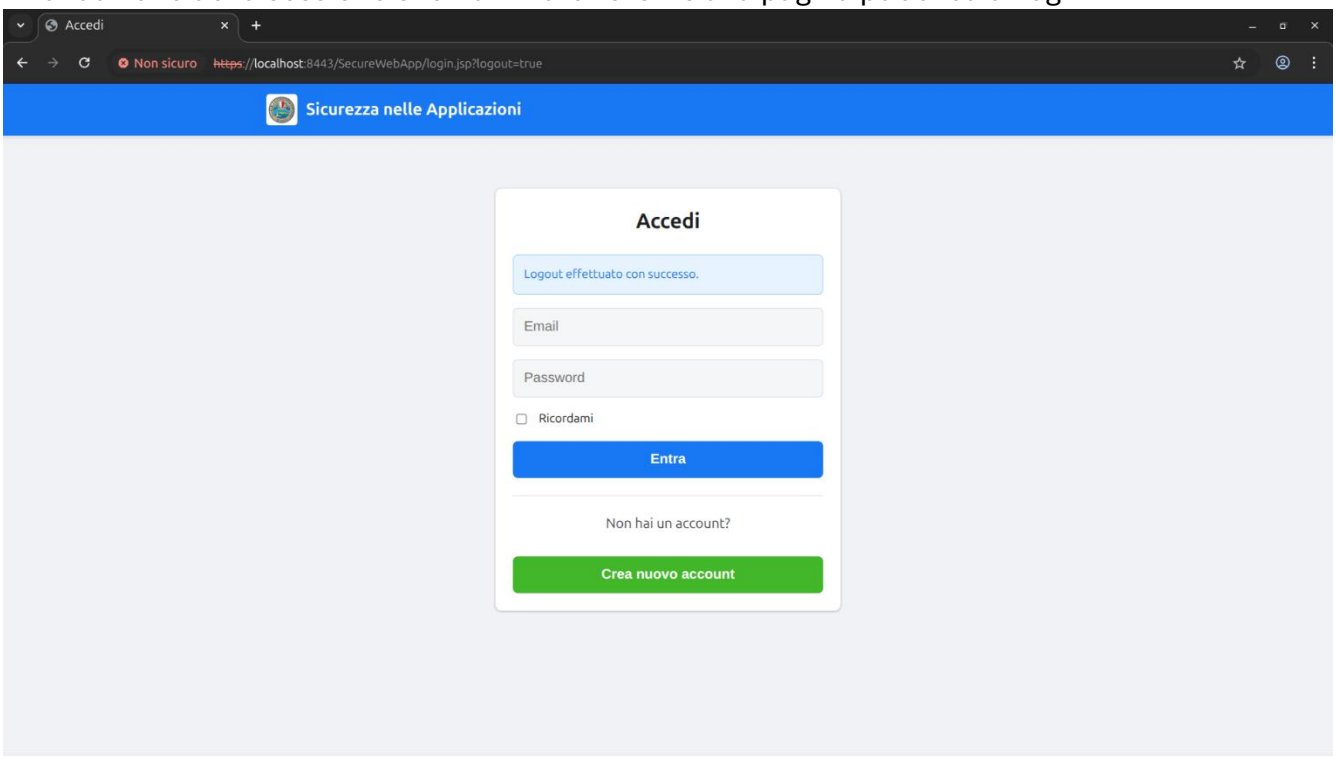
Comportamento osservato: Dopo il periodo di inattività, al primo tentativo di navigazione, l'applicazione ha reindirizzato l'utente alla schermata di login.

3.1.8 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.
Input fornito:

- L'utente, precedentemente autenticato e presente nella pagina ***home.jsp***, clicca sul pulsante "Esci" nella barra di navigazione.

Comportamento atteso: Il server deve ricevere la richiesta di logout, invocare il metodo di invalidazione della sessione e reindirizzare l'utente alla pagina pubblica di login.



Comportamento osservato: Dopo il click su "Esci", il sistema ha reindirizzato immediatamente alla pagina ***login.jsp***.

3.1.9 TU9 - Tentativo di accesso post-logout

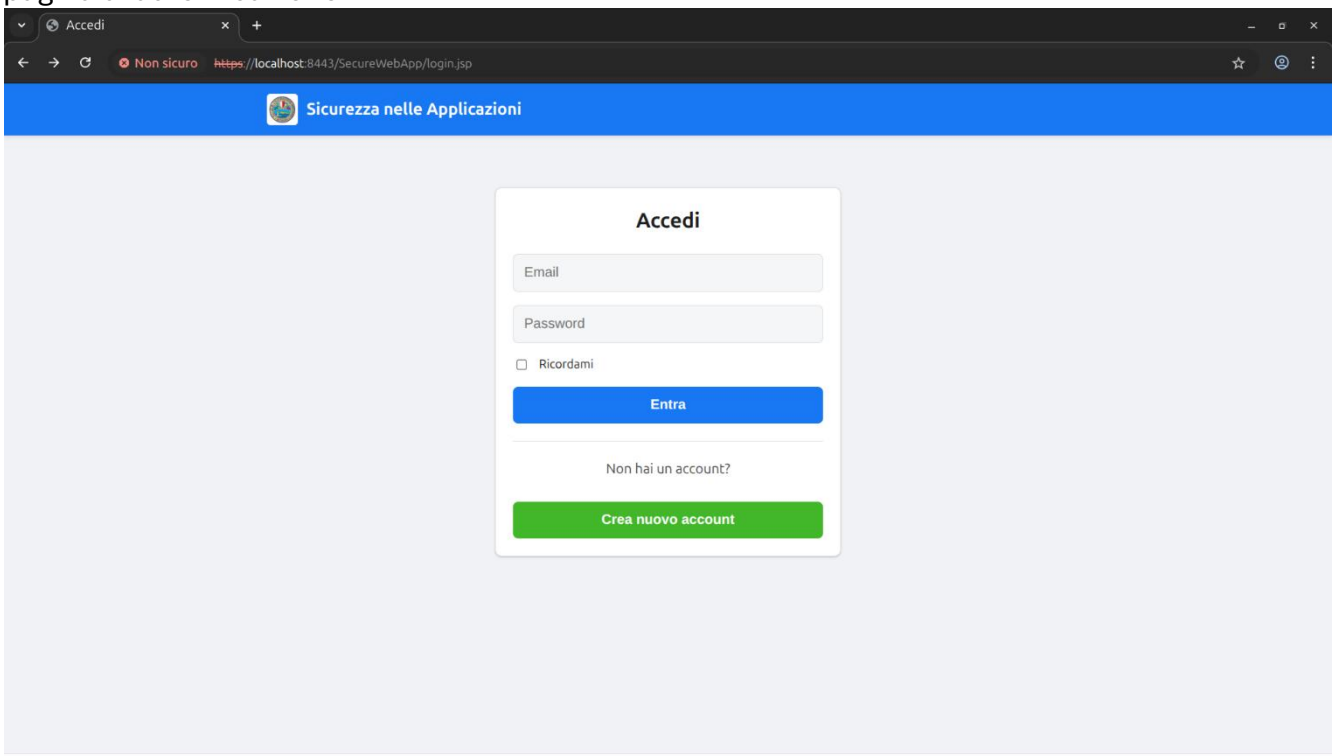
Verifica che, dopo il logout, un tentativo di accesso a risorse riservate venga correttamente bloccato dal sistema.

Input fornito:

- L'utente ha appena effettuato il Logout e si trova sulla pagina di Login.
- L'utente preme il pulsante "Indietro" del browser e preme Invio.

Comportamento atteso: Il server deve rilevare che la richiesta proviene da un client senza una sessione valida. Di conseguenza, deve bloccare il caricamento della pagina protetta e reindirizzare nuovamente l'utente alla pagina ***login.jsp***.

Comportamento osservato: Tentando di accedere all'URL riservato dopo il logout, l'applicazione ha impedito la visualizzazione e ha reindirizzato immediatamente il browser alla pagina di autenticazione.



3.1.10 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.

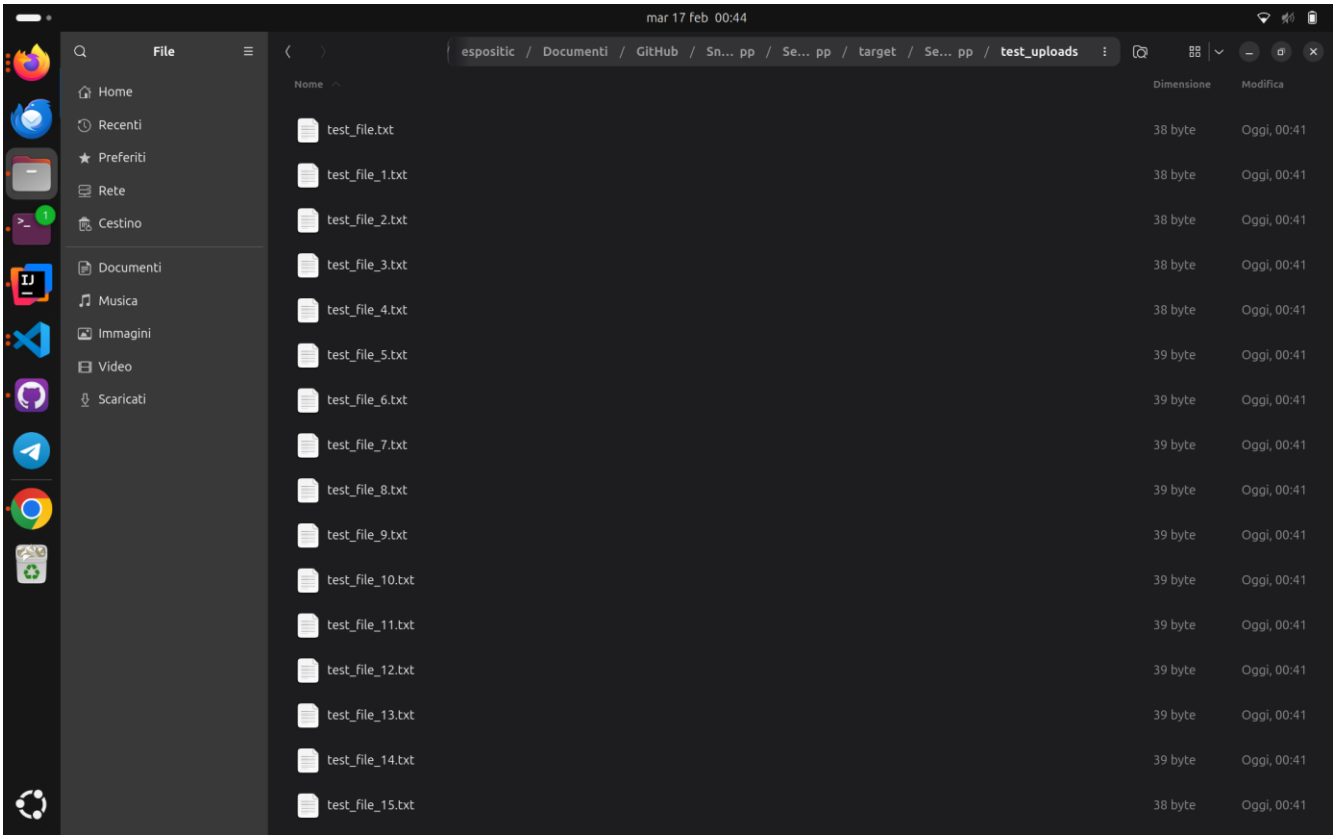
Input fornito:

- Esecuzione della servlet di test dedicata ***/test-concurrency***.
- La servlet istanzia un pool di 20 Thread concorrenti.
- Ciascuno dei 20 thread tenta simultaneamente di salvare un file con lo stesso nome originale: test_file.txt, invocando il metodo condiviso ***FileManagerUtil.saveFileSafe()***.

Comportamento atteso: Il meccanismo di sincronizzazione deve serializzare l'accesso al blocco di codice che controlla l'esistenza del file e decide il nome.
Nonostante le richieste siano simultanee, il sistema deve:

- Rilevare che documento.txt esiste già per il secondo thread.
- Generare un nome univoco progressivo.
- Garantire che alla fine del test siano presenti su disco esattamente 20 file distinti, senza alcuna perdita di dati.

Comportamento osservato: Al termine dell'esecuzione, la pagina di report ha confermato che tutti i 20 thread hanno completato l'operazione. Verificando la directory concurrent_uploads, sono stati trovati esattamente 20 file.



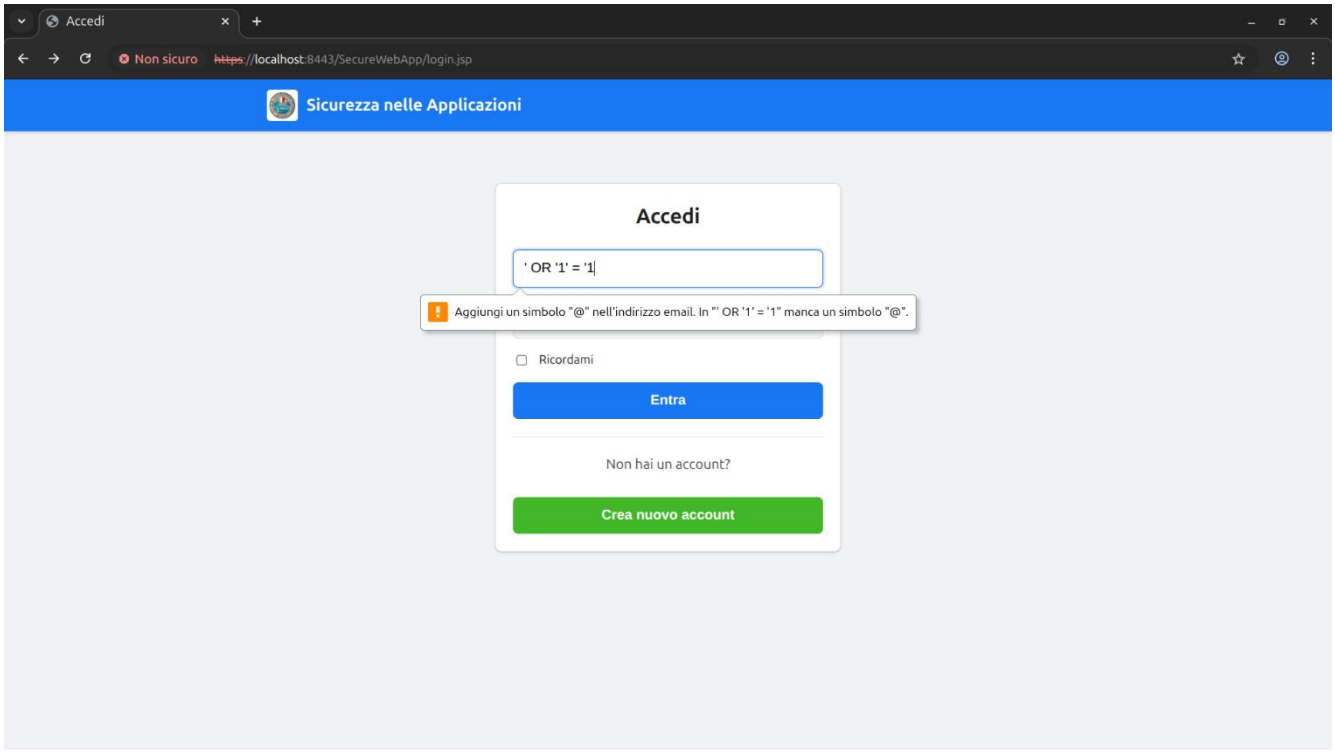
3.2 Test di abuso

3.2.1 TA1 Tentativo di SQL Injection nel campo e-mail del login

Verificare che il modulo di autenticazione sia immune a tentativi di manipolazione delle query SQL, impedendo a un attaccante di bypassare il controllo password o accedere come un altro utente tramite l'inserimento di caratteri speciali nel campo e-mail.
Input fornito:

- Campo E-mail: ' OR '1'='1
- Campo Password: Qualsiasi valore, dato che l'obiettivo è bypassare il controllo sull'email.

Comportamento atteso: Il sistema deve trattare la stringa ' OR '1'='1 come un semplice valore letterale e non come codice SQL eseguibile.
Comportamento osservato: L'applicazione gestisce l'input, restituendo il messaggio di errore standard "Aggiungi un simbolo “@" nell’indirizzo e-mail in ' OR '1'='1 manca un simbolo “@””. L'accesso è stato negato e non sono state rivelate informazioni sulla struttura del database.



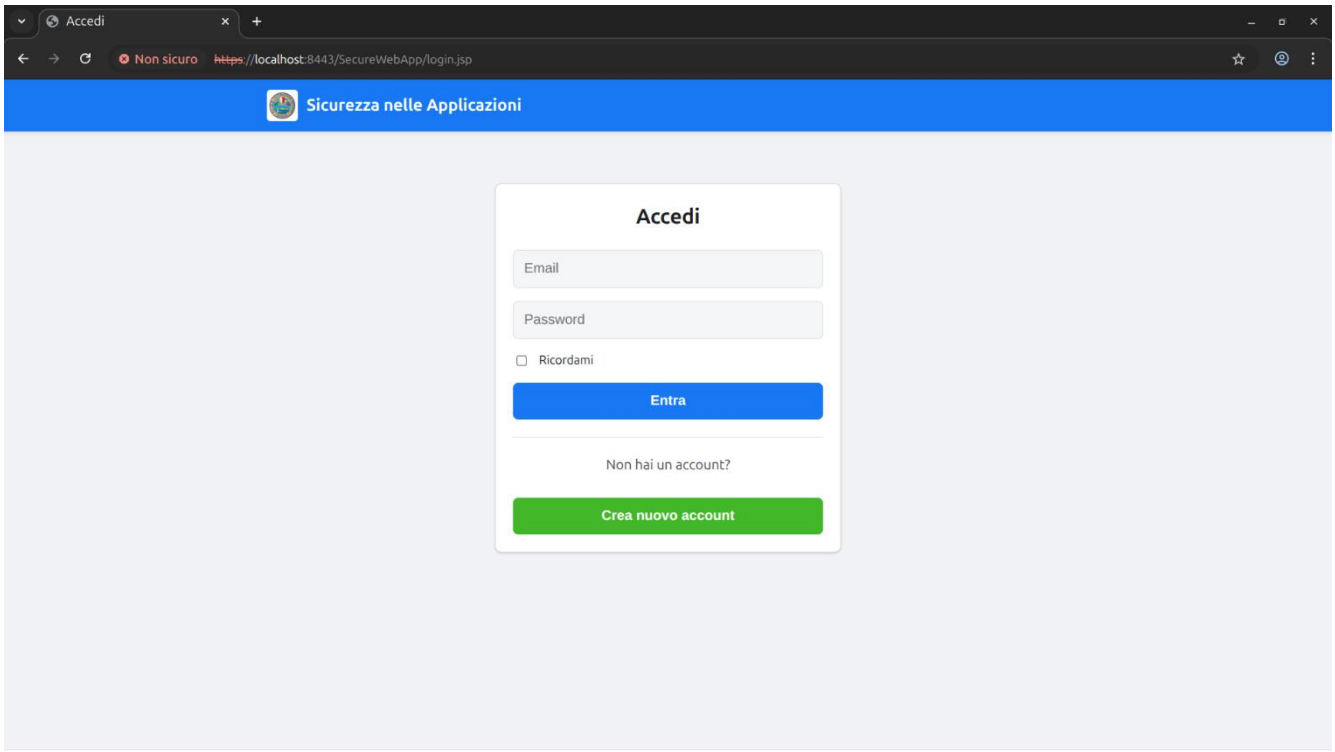
Contromisura applicata: Al fine di mitigare attacchi di tipo SQL Injection, l’applicazione gestisce verifica che la mail sia conforme allo standard grazie alla classe ***ValidationUtil***, e successivamente la query effettuata sul db è effettuata attraverso le ***java.sql.PreparedStatement,*** abbiamo quindi una doppia barriera per mitigare questa tipologia di attacchi. In questo caso l’attacco non è nemmeno arrivato al server perché i moderni browser effettuano una verifica sui campi e-mail nei form ancor prima che l’attacco arrivi al server.

3.2.2 TA2 Tentativo di bypass dell'autenticazione

Verificare che le risorse riservate (es. ***home.jsp, feed.jsp***) siano protette da tentativi di accesso diretto tramite URL, impedendo a utenti non autenticati o anonimi di visualizzare contenuti sensibili senza passare dalla procedura di login.
Input fornito:

- L'utente non autenticato digita manualmente nella barra degli indirizzi del browser l'URL completo di una risorsa protetta, ad esempio: https://localhost:8443/SecureWebApp/home.jsp e preme Invio.

Comportamento atteso: Il server deve intercettare la richiesta prima di renderizzare la pagina JSP. Deve verificare l'assenza di una sessione valida e bloccare l'accesso, reindirizzando immediatamente il client alla pagina ***login.jsp***. Non deve essere mostrato alcun contenuto della pagina richiesta, nemmeno per una frazione di secondo.
Comportamento osservato: Nonostante l'URL inserito puntasse alla pagina riservata, il browser è stato reindirizzato automaticamente alla pagina di Login.



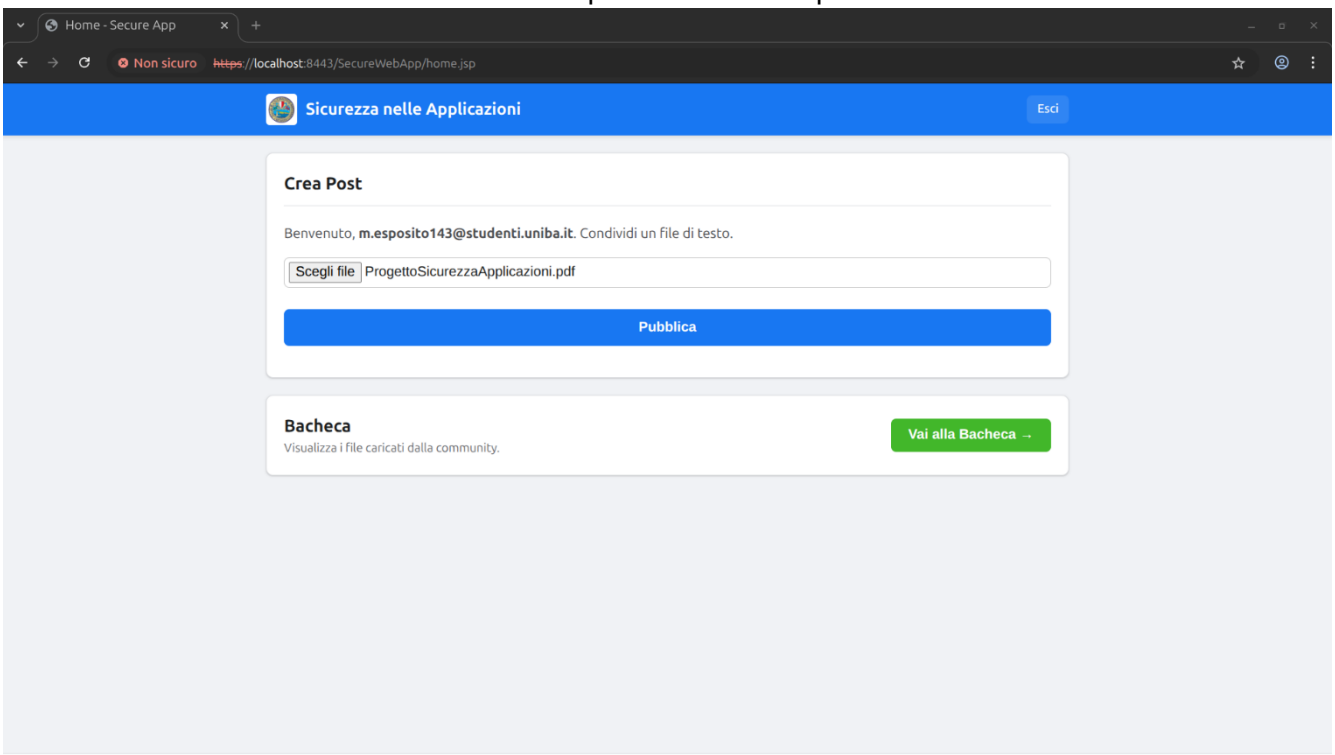
Contromisura applicata: È stato implementato un filtro servlet ***AuthenticationFilter*** mappato sulle risorse sensibili. Questo garantisce che la verifica dei permessi avvenga a monte di qualsiasi logica di visualizzazione.

3.2.3 TA3 Upload di file con estensione vietata

Verificare che il meccanismo di validazione dei file rifiuti categoricamente l'upload di file con estensioni o formati non presenti nella whitelist approvata, proteggendo il server dall'esecuzione di codice arbitrario.

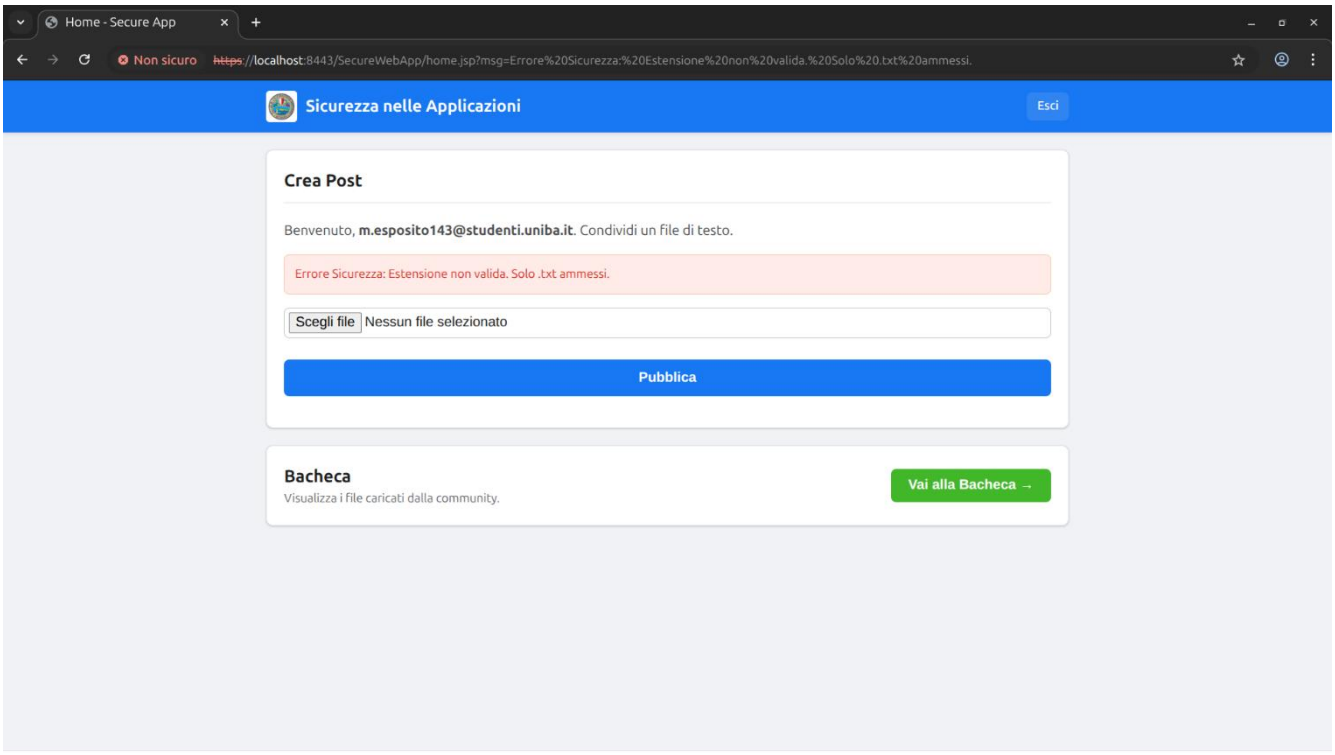
Input fornito:

- File denominato ProgettoSicurezzaApplicazioni.pdf.
- Selezione del file tramite il form di upload e click sul pulsante "Pubblica".



Comportamento atteso: Il sistema deve analizzare il file prima del salvataggio definitivo. Rilevando che l'estensione non corrisponde a .txt, l'applicazione deve interrompere l'operazione, non salvare nulla sul disco e restituire un messaggio di errore chiaro all'utente.

Comportamento osservato: Dopo aver tentato l'upload del file proibito, l'applicazione ha ricaricato la pagina mostrando un messaggio di errore rosso: "Estensione non valida. Solo .txt ammessi". Verificando la directory di upload sul server, il file non è stato caricato.



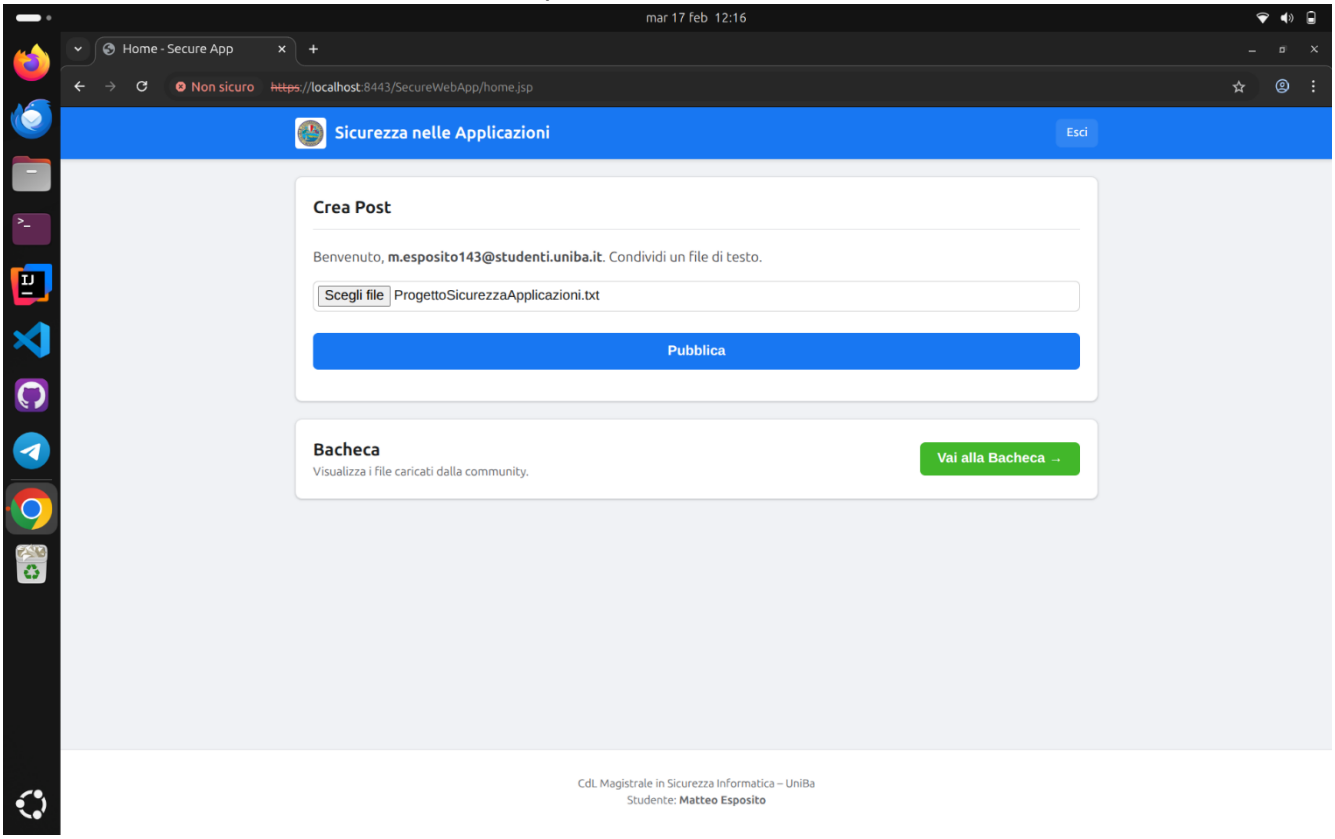
Contromisura applicata: Nella logica di ***UploadServlet*** è implementato un controllo basato su una lista di permessi espliciti. Il file non passa il controllo sul primo filtro sull'estensione .txt.

3.2.4 TA4 Upload di file con estensione lecita ma contenuto malevolo

Verificare che il sistema analizzi il contenuto reale del file e non si fidi esclusivamente dell'estensione fornita dall'utente. Un attaccante non deve poter caricare un eseguibile o un file binario semplicemente rinominandolo con estensione .txt.

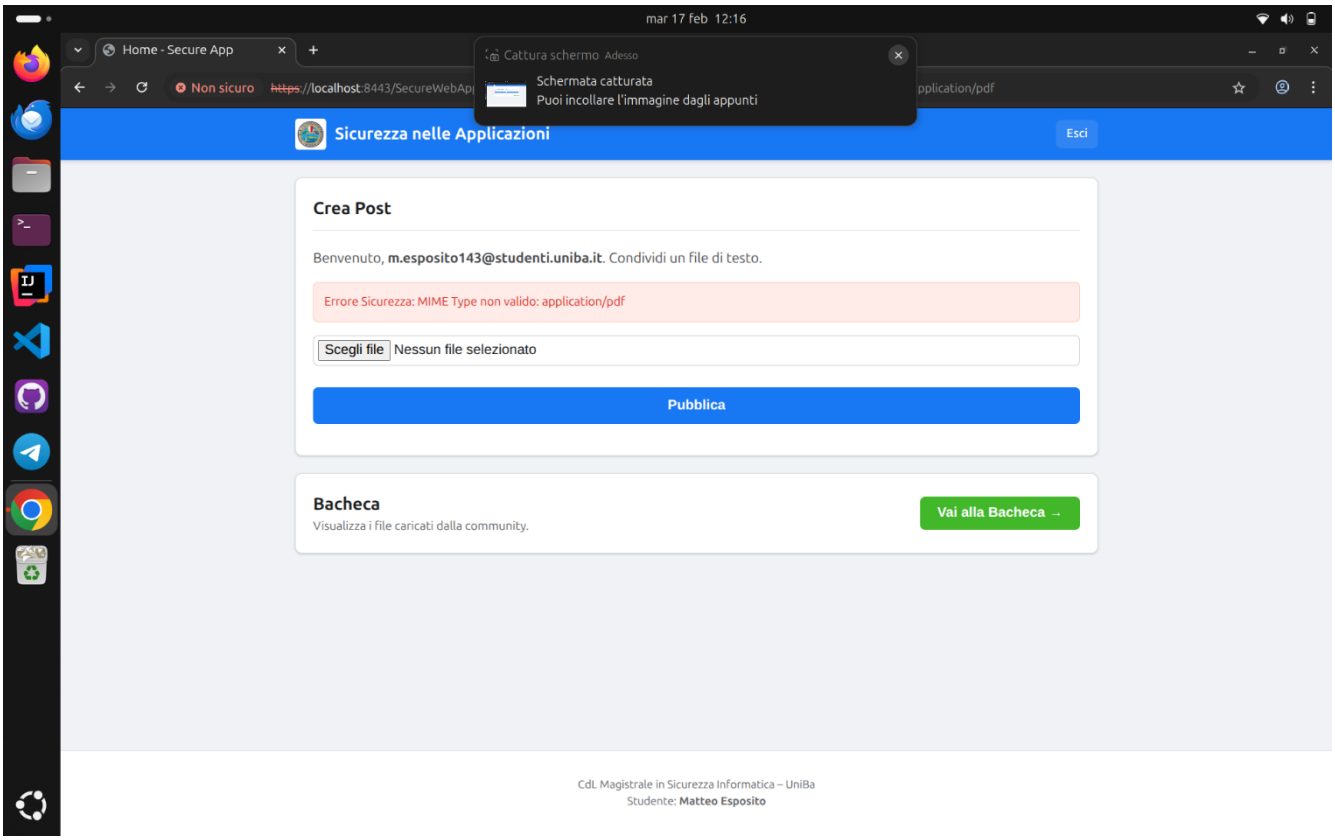
Input fornito:

- Si prende un file pdf ***ProgettoSicurezzaApplicazioni.pdf***.
- Si rinomina manualmente il file in ***ProgettoSicurezzaApplicazioni.txt***.
- Si carica il file tramite il form di upload, che teoricamente accetta i file .txt.



Comportamento atteso: Il sistema deve leggere i Magic Numbers del file.
Deve rilevare che, nonostante il nome finisca per .txt, l'intestazione del file corrisponde a un pdf. Conseguentemente, deve bloccare il caricamento e restituire un errore di sicurezza, impedendo che il file corrotto venga salvato sul server.

Comportamento osservato:
L'applicazione ha restituito un messaggio di errore "MIME Type non valido: application/pdf".
Verificando sul server, il file ***ProgettoSicurezzaApplicazioni.txt*** non è stato salvato.



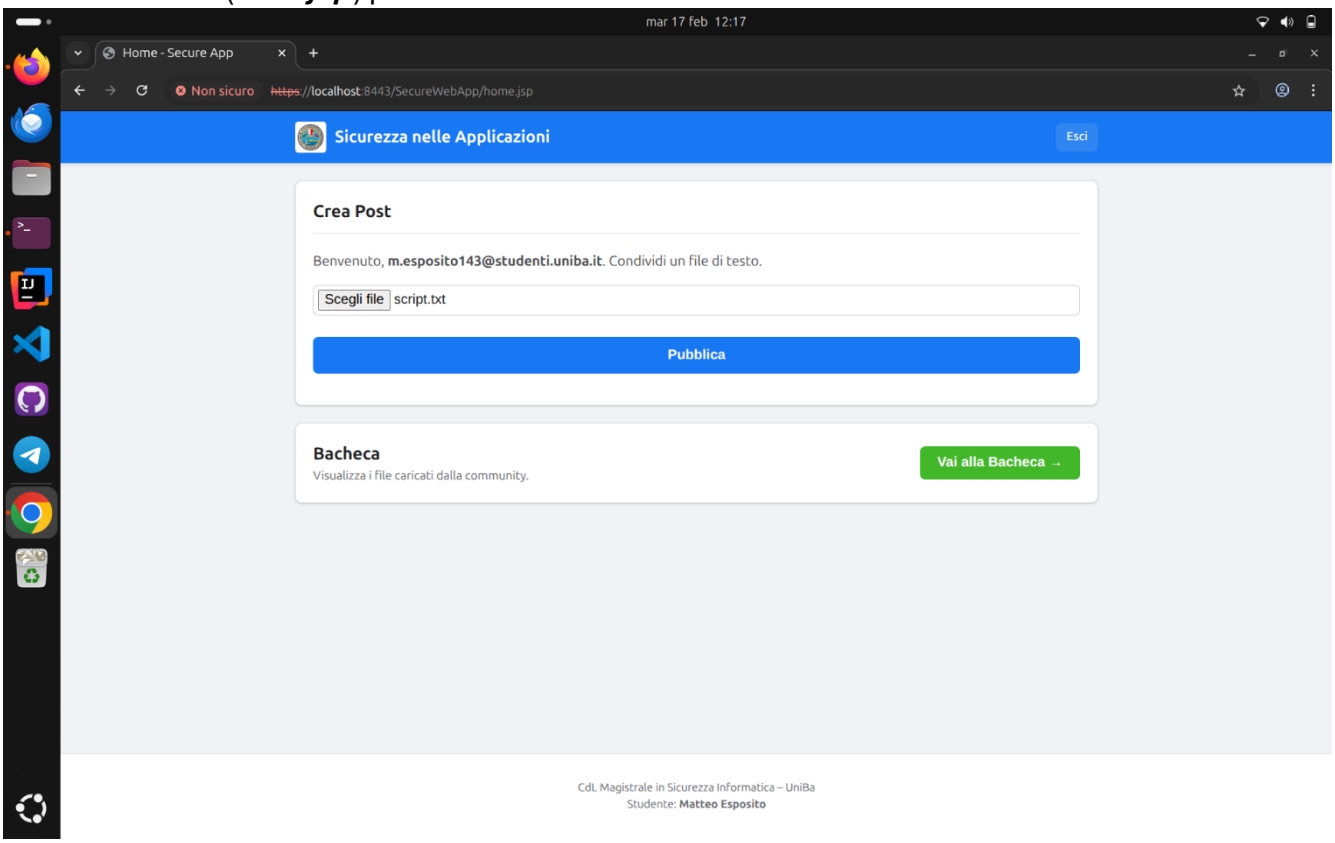
Contromisura applicata: Utilizzo della libreria ***Apache Tika*** all'interno della ***UploadServlet***. Il controllo ***endsWith(".txt")*** non basta, è servito analizzare lo stream di byte in ingresso

3.2.5 TA5 Upload di contenuto testuale con script per stored XSS

Verificare che il sistema sia resiliente agli attacchi di tipo Stored XSS. L'obiettivo è confermare che, anche se un attaccante riesce a caricare un file di testo contenente codice JavaScript malevolo, questo codice venga neutralizzato in fase di visualizzazione e non venga mai eseguito dal browser delle vittime.

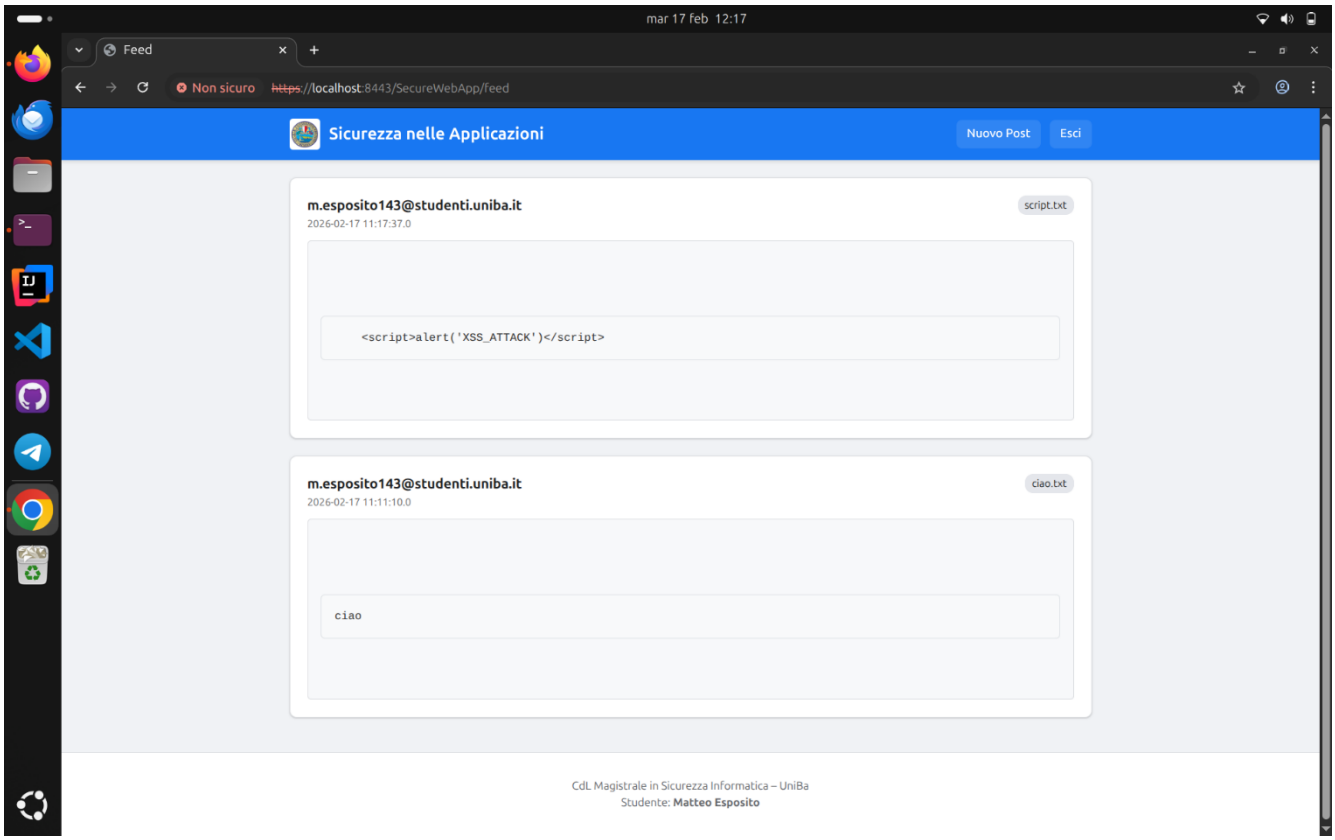
Input fornito:

- Un file di testo script.txt contenente il seguente vettore d'attacco: ***<script>alert('XSS_ATTACK')</script>***
- Caricamento del file tramite la funzionalità di upload e successiva navigazione nella bacheca (***feed.jsp***) per visualizzarlo.



Comportamento atteso: Il sistema accetterà il file, ma al momento del rendering HTML, i caratteri speciali (<, >, &, ") devono essere convertiti nelle rispettive Entità HTML (<, >, ecc.). Il browser deve visualizzare il codice sorgente dello script come semplice testo.

Comportamento osservato: Il file è stato caricato correttamente. Accedendo alla pagina dei post, il payload è stato visualizzato a schermo come ***<script>alert('XSS_ATTACK')</script>.*** Il codice JavaScript non è stato interpretato dal browser e l'attacco è stato neutralizzato.



Contromisura applicata: L'applicazione utilizza la tag library **JSTL** (JavaServer Pages Standard Tag Library) per l'escape dell'output. L'istruzione ***<c:out value="${p.content}" />*** scansiona il contenuto recuperato dal database e trasforma automaticamente tutti i metacaratteri HTML in entità sicure prima di inviarli al client, annullando l'efficacia di qualsiasi script iniettato.

3.2.6 TA6 Accesso a risorse protette senza sessione valida

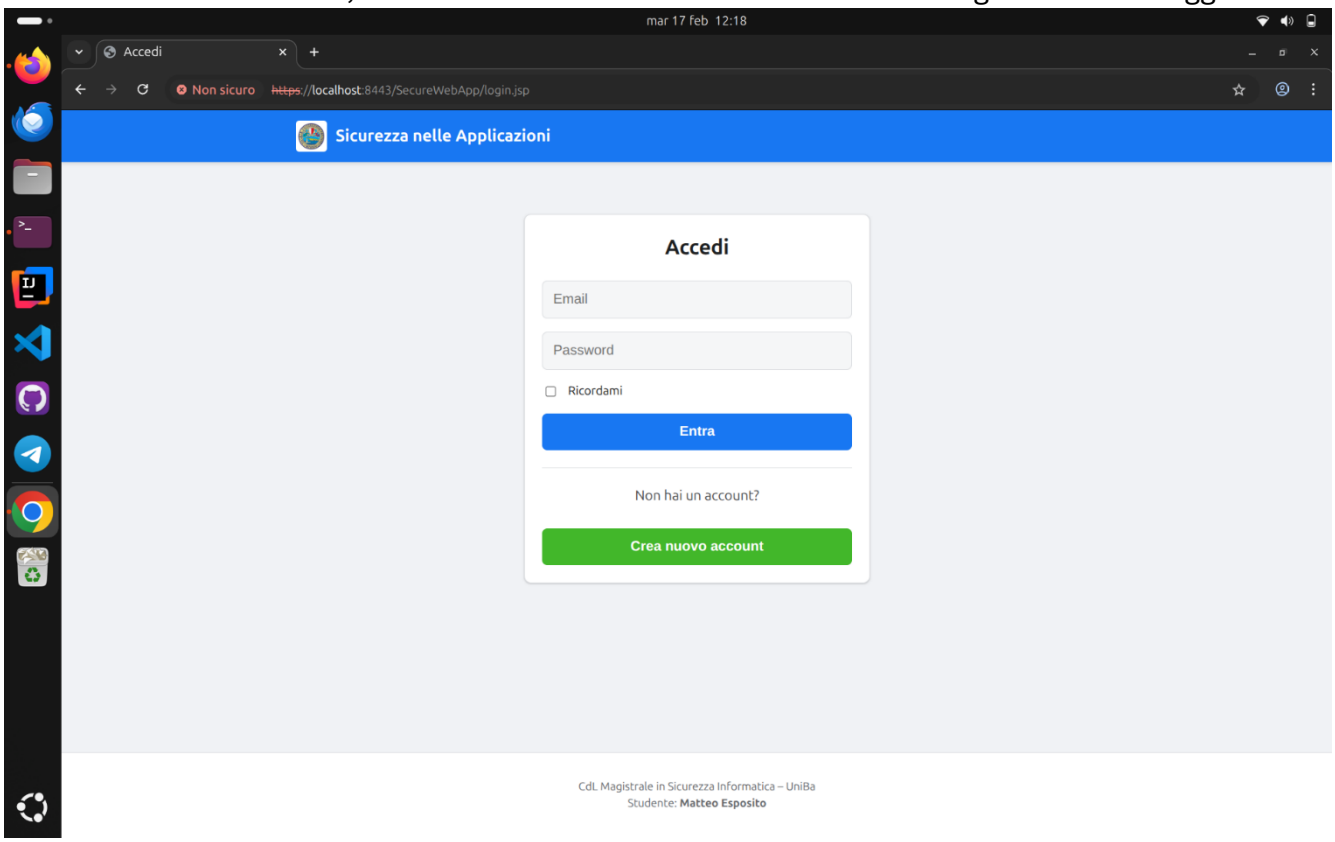
Verificare che il filtro di autenticazione sia attivo globalmente su tutte le risorse riservate e che impedisca l'accesso a chiunque non presenti un cookie valido. Questo simula un attaccante che tenta di accedere direttamente alle pagine interne senza mai essere passato dal login.

Input fornito:

- L'attaccante ottiene l'URL diretto di una pagina riservata (es. https://localhost:8443/SecureWebApp/feed.jsp). Apre una finestra di navigazione dove non esistono cookie salvati e incolla l'URL nella barra degli indirizzi.

Comportamento atteso: Tomcat deve intercettare la richiesta HTTP, ***AuthenticationFilter*** deve verificare l'esistenza della sessione utilizzando ***request.getSession(false)***, il server deve forzare un redirect verso la pagina ***login.jsp***.

Comportamento osservato: Il browser, pur avendo l'URL della pagina ***feed.jsp*** nella barra degli indirizzi, ha caricato visualmente la pagina di Login. Non è stato possibile visualizzare alcun contenuto della bacheca, confermando che la risorsa è inaccessibile agli utenti non loggati.



Contromisura applicata: L'applicazione utilizza un filtro che agisce come un gateway, intercetta tutte le richieste verso le risorse protette, se manca l'oggetto User in sessione, blocca la catena di esecuzione (***chain.doFilter***) reindirizzando al login.

3.2.7 TA7 Riutilizzo di cookie o sessione scaduta

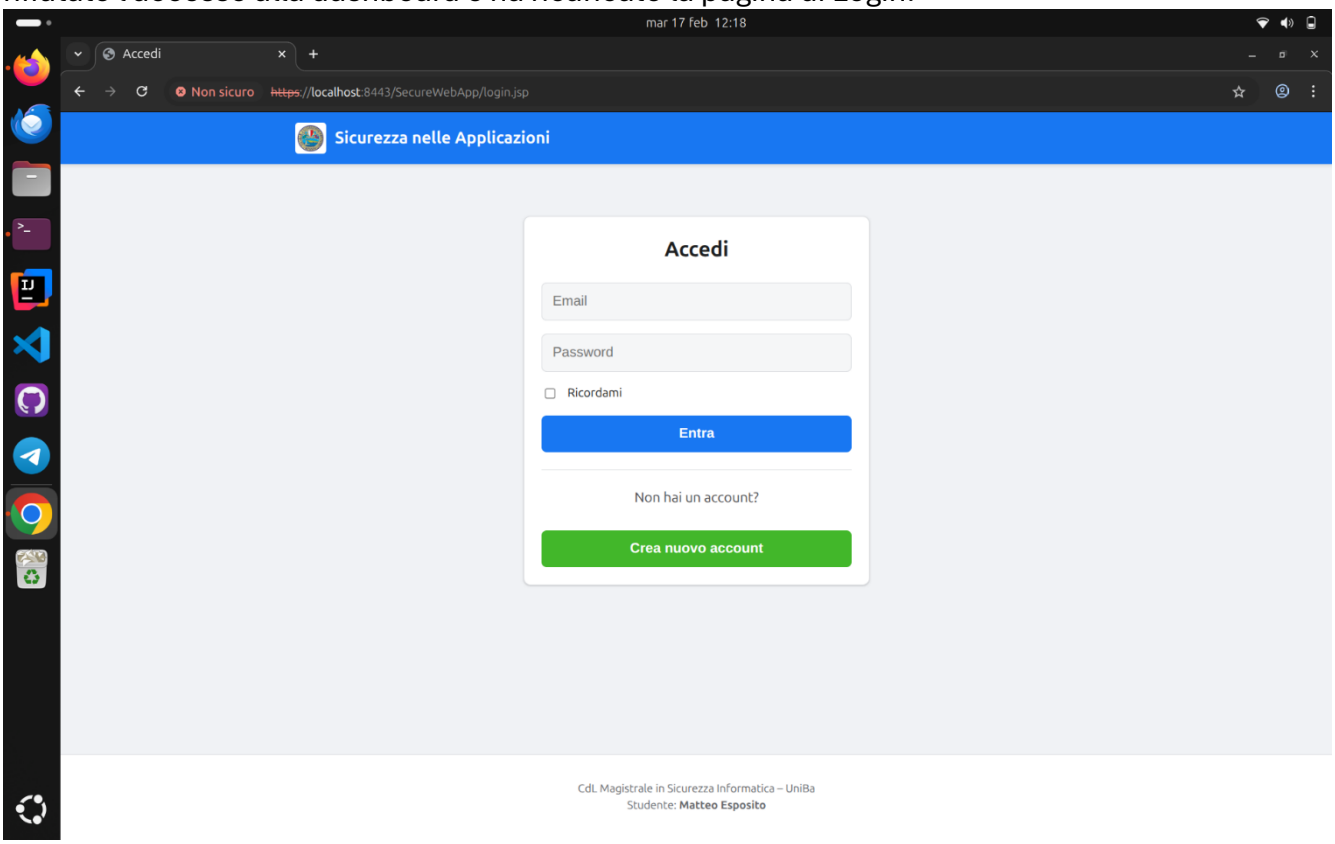
Verificare che un identificativo di sessione JSESSIONID precedentemente valido, ma appartenente a una sessione ormai scaduta, non possa essere riutilizzato da un attaccante per ottenere l'accesso non autorizzato.

Input fornito:

- Login legittimo. Si apre la console del browser, si copia il valore del JSESSIONID corrente e lo si salva in un blocco note.
- Si effettua il Logout. La sessione sul server viene distrutta.
- Dalla pagina di login, si apre nuovamente la console (F12), si modifica manualmente il valore del cookie JSESSIONID attuale incollando quello salvato e si tenta di forzare l'accesso andando su http://localhost:8443/SecureWebApp/home.jsp.

Comportamento atteso: Il server riceve la richiesta con il vecchio ID. Cerca nella sua memoria interna l'oggetto HttpSession corrispondente a quell'ID. Non trovandolo, deve trattare la richiesta come non autenticata. L'utente deve essere immediatamente reindirizzato alla pagina di login, rendendo inutile il cookie rubato.

Comportamento osservato: Nonostante il browser abbia inviato il vecchio cookie, il server ha rifiutato l'accesso alla dashboard e ha ricaricato la pagina di Login.



Contromisura applicata: La validità della sessione è determinata esclusivamente da Tomcat, non dal client. Il cookie è solo un riferimento. Quando viene invocato ***session.invalidate()*** o scade il timeout, l'oggetto sessione viene rimosso dalla memoria del server. Qualsiasi richiesta successiva con quell'ID viene considerata orfana e scartata, mitigando il rischio di Session Fixation o riutilizzo.

3.2.8 TA8 Tentativo di esecuzione di file caricati

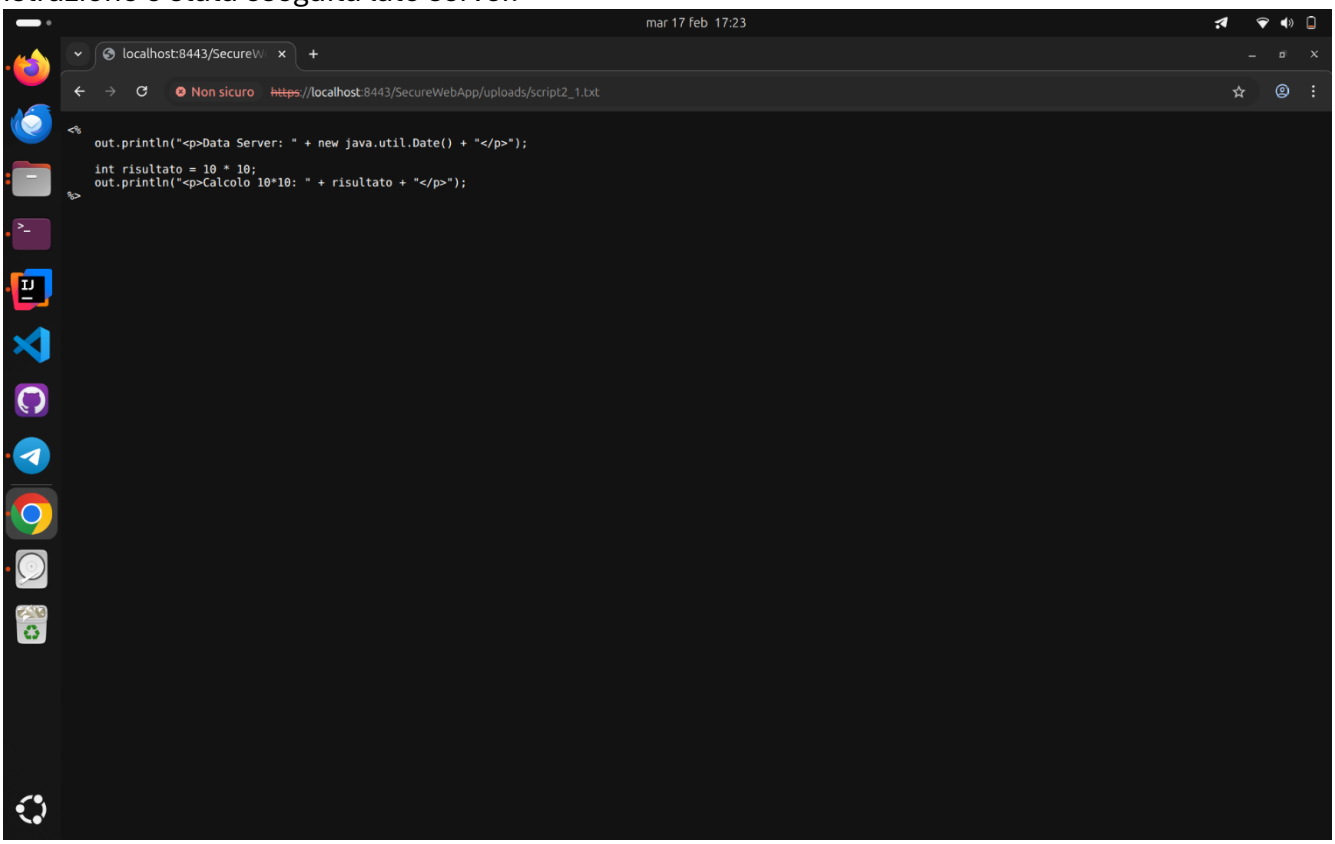
Verificare che i file caricati nella directory di upload non possano essere eseguiti come script o programmi dal server web. Questo serve a prevenire attacchi di tipo RCE (Remote Code Execution), dove un attaccante carica una shell mascherata e tenta di eseguirla richiamandone l'URL.

Input fornito:

- Un file di testo denominato script2.txt contenente codice potenzialmente eseguibile.
- Caricamento del file tramite il form.
- Tentativo di accesso diretto al file tramite URL (es. https://localhost:8443/SecureWebApp/upload/script2.txt).

Comportamento atteso: Il server deve restituire il file esclusivamente come contenuto statico (Content-Type: text/plain). Il codice contenuto all'interno non deve essere processato dal motore JSP/Servlet di Tomcat. Il browser deve limitarsi a mostrare il testo del codice sorgente, senza eseguirlo.

Comportamento osservato: Navigando direttamente all'URL del file caricato, il browser ha visualizzato il contenuto testuale del file (il codice sorgente) come semplice testo. Nessuna istruzione è stata eseguita lato server.



Contromisura applicata:

È stato attivato il filtro nativo di Tomcat HttpHeaderSecurityFilter all'interno del file di configurazione web.xml. Questo filtro inietta automaticamente l'header HTTP X-Content-Type-Options: nosniff in tutte le risposte del server.

Il server è configurato per servire i file caricati nella directory di upload esclusivamente con Content-Type statici e sicuri (come text/plain o i formati immagine validati), impedendo che vengano serviti come text/html, application/javascript o altri formati eseguibili.