Università degli Studi di Udine

Corso di Cybersecurity

Anno 2023/204

Gianluca Piarulli 165824

Soluzioni di Cybersecurity per Piattaforme Online

Introduzione

Connessioni Creative è una web magazine nata dalla mente di alcuni studenti del corso di Editoria Digitale, per l'esattezza è uno spazio digitale dedicato all'esplorazione e alla reinterpretazione degli impatti profondi che la pandemia di COVID-19 ha avuto sugli individui, sulle comunità e sul mondo dell'arte.
Il mio obiettivo è sviluppare l'applicazione web e il database relazionale in cui gli utenti possano salvare il magazine o i singoli articoli per la lettura, mettendo tutto l'ecosistema in sicurezza.

Architettura del Server Web

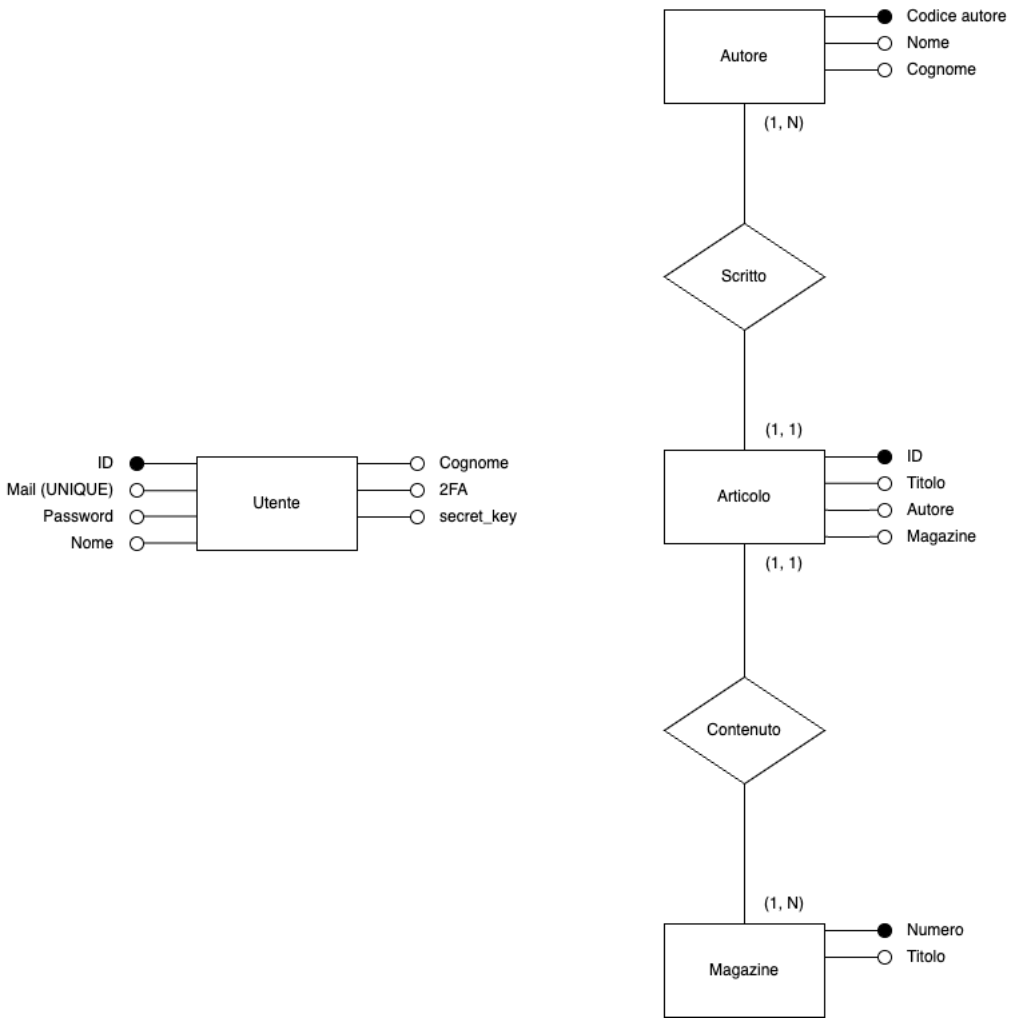
1. **Server Web (Apache)**:
   - **Descrizione**: Utilizzo Apache come server web per gestire le richieste HTTPS. Questo server è responsabile della ricezione delle richieste dagli utenti, della gestione delle connessioni e della fornitura delle risposte appropriate.
   - **Funzionalità**: Include il bilanciamento del carico, la gestione dei certificati SSL/TLS per le connessioni sicure e il supporto per i moduli di autenticazione.
2. **Linguaggio di Programmazione (PHP, JS, HTML, CSS)**:
   - **Descrizione:** Il lato server è stato sviluppato utilizzando PHP, il quale interagisce direttamente con il database. Il lato client è stato sviluppato con HTML, CSS e Javascript per gestire eventi e richieste AJAX.
   - **Funzionalità:** Esegue la logica applicativa, gestisce le sessioni utente, elabora i dati e interagisce con il database.
3. **Database (MySQL)**:
   - **Descrizione**: Utilizzo MySQL come sistema di gestione di database relazionali per la memorizzazione e la gestione dei dati.
   - **Funzionalità**: Gestisce le tabelle, esegue query SQL e offre un'interfaccia tramite phpMyAdmin per semplificare l'amministrazione e la manutenzione dei dati.
4. **Interfaccia di Gestione (phpMyAdmin)**:
   - **Descrizione**: phpMyAdmin è utilizzato per l'amministrazione del database MySQL.
   - **Funzionalità**: Permette la gestione di database, tabelle, colonne, relazioni, indici, utenti e permessi. Supporta l'importazione e l'esportazione dei dati, così come la costruzione di query complesse tramite un'interfaccia grafica intuitiva.

5. **Sicurezza (SSL/TLS)**:
    - **Descrizione**: Ho implementato misure di sicurezza come certificati SSL/TLS, autenticazione a due fattori, CAPTCHA, etc. per proteggere il server web e i dati degli utenti.
    - **Funzionalità**: Il certificato SSL/TLS protegge le comunicazioni tra il server e i client crittografando i dati trasferiti. L'autenticazione a due fattori richiede agli utenti di fornire un secondo metodo di verifica oltre alla password, aumentando la sicurezza dell'accesso. Il CAPTCHA aiuta a prevenire l'accesso automatizzato da parte di bot o software dannosi, assicurando che le richieste siano generate da utenti legittimi.

Database

Il database relazionale è stato implementato con MySQL e gestito tramite l'interfaccia di phpMyAdmin.

1. Modello concettuale



2. Modello logico

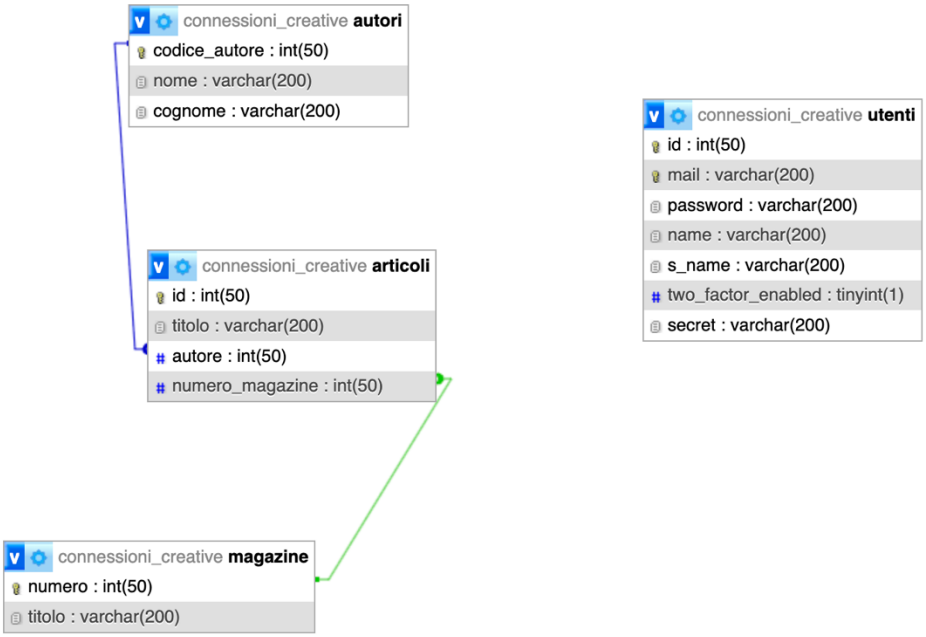
Utente (id (int), mail (varchar, UNIQUE), nome (varchar), cognomen (varchar), password (varchar), 2FA (tinyint), secret_key (varchar))

Magazine (numero (int), titolo (varchar))

Articolo (id (int), numero_magazine (int), titolo (varchar), autore (varchar))

Autore (codice_autore (int), nome (varchar), cognome (varchar))

3. Design Relazionale Database



Main features

Certificato SSL/TLC

Per garantire la riservatezza e l'integrità dei dati trasmessi tra il server e i client è necessario implementare un certificato SSL/TLS per il nostro server Apache.

Questa pratica è utile per diversi motivi:

1. **Protezione dei Dati Sensibili**:
   L'utilizzo di un certificato SSL/TLS cripta i dati trasmessi tra il server e i client, rendendo difficile per gli attaccanti intercettare o alterare le informazioni.

2. **Autenticazione del Server**:
   Un certificato SSL/TLS fornisce un meccanismo per autenticare l'identità del server. I client possono verificare che il server a cui si stanno collegando è effettivamente quello dichiarato, prevenendo attacchi di phishing e man-in-the-middle.

3. **Integrità dei Dati**:
   Un certificato SSL/TLS garantisce che i dati trasmessi non siano stati alterati durante il transito. Questo protegge sia il server che i client da attacchi che potrebbero manipolare le informazioni in transito.

Per utilizzare un certificato SSL/TLS, un sito web deve avere un certificato SSL valido installato sul server web. Questo certificato è emesso da una Certificate Authority (CA) e contiene informazioni crittografiche che consentono al server di stabilire una connessione sicura con i browser/clienti.
Nel nostro caso fingeremo di essere noi la CA, genereremo e firmeremo il certificato attraverso tools specifici del pacchetto OpenSSL contenuto in Apache.

Il nostro certificato avrà al suo interno:

1. **Chiave pubblica del server**: utilizzata per crittografare i dati durante la comunicazione con il client.
2. **Identità del proprietario del certificato**: il nome del proprietario o dell'organizzazione che possiede il certificato.
3. **Chiave pubblica del proprietario**: utilizzata dal client per decrittografare i dati crittografati dalla chiave pubblica del server.
4. **Firma digitale di una Certificate Authority (CA)**: Un certificato SSL è firmato digitalmente da una CA attendibile. Questa firma garantisce che il certificato sia autentico e proviene da una fonte attendibile.
5. **Data di validità**: data di inizio e fine validità, poiché un certificato SSL ha una durata limitata.

Protezione del database

Il database contiene informazioni sensibili e preziose, come dati degli utenti. Verrà implementata una password forte e sicura per impedire l'accesso non autorizzato a questi dati da parte di individui o software dannosi.

Sicurezza per Registrazione e Login

1. **Lato server**

Gestione degli errori:
- Tutte le funzioni gestiscono le eccezioni PDO (PDOException) in modo da non mostrare dettagli sensibili agli utenti finali.
- L'uso di error_log() registra gli errori in un file di log sicuro, permettendo agli amministratori di monitorare e risolvere i problemi senza esporre informazioni sensibili agli utenti.

Connessione al database sicura:

- La funzione connectToDatabase() utilizza PDO con l'opzione PDO::ERRMODE_EXCEPTION per gestire le eccezioni in modo robusto.
- Le credenziali di accesso al database ($servername, $dbusername, $dbpassword, $dbname) sono mantenute in modo sicuro e non sono esposte direttamente nel codice visibile agli utenti.

Sanitizzazione e validazione dei dati:

- I dati ricevuti dai form ($_POST) vengono sanitizzati e validati per prevenire attacchi di tipo injection o inserimento di dati dannosi.
- Esempi includono l'uso di filter_var() per l'indirizzo email e la funzione htmlspecialchars() per evitare XSS (Cross-Site Scripting).

Utilizzo di prepared statements:

- Le query SQL sono preparate utilizzando prepared statements ($conn->prepare()) per prevenire SQL injection, una delle vulnerabilità più comuni nelle applicazioni web.

Gestione delle sessioni:

- Le funzioni registration() e login() gestiscono le sessioni in modo sicuro, utilizzando session_start() e assegnando l'ID dell'utente alla sessione solo dopo la verifica delle credenziali.

Protezione contro bot e attacchi di forza bruta:

- Nella funzione registration(), l'uso di reCAPTCHA con verifica del token aiuta a prevenire registrazioni automatizzate e attacchi di tipo bot.
- Il reCAPTCHA è un servizio di Google progettato per distinguere tra utenti umani e bot automatizzati durante le interazioni su siti web. Questo strumento è ampiamente utilizzato per prevenire accessi non autorizzati e attività dannose generate da programmi automatizzati. Funziona presentando agli utenti test che possono essere facilmente risolti da persone reali ma risultano complessi o impossibili per i bot. In questo modo, il reCAPTCHA contribuisce significativamente a migliorare la sicurezza online, proteggendo i dati degli utenti e garantendo interazioni web più affidabili e autentiche.

Riduzione delle informazioni di debug:

- L'uso di messaggi generici di errore ("Connessione fallita.", "Errore durante il recupero dei dati.", etc.) protegge l'utente finale da vedere dettagli specifici degli errori che potrebbero essere utilizzati da attaccanti per scoprire vulnerabilità.

Chiusura sicura delle connessioni:

- Ogni funzione chiude la connessione al database ($conn = null;) usando finally per assicurarsi che le risorse siano liberate correttamente e per prevenire potenziali attacchi di tipo DoS (Denial of Service) attraverso la saturazione delle connessioni.

2. **Lato client**

Validazione dei campi del modulo:

- Utilizzo JavaScript per garantire che i campi del modulo (come nome, cognome, email e password) siano correttamente compilati prima di essere inviati al server. Questo aiuta a prevenire che dati non validi o dannosi vengano inviati al backend, riducendo il rischio di vulnerabilità.

Controlli avanzati sulla password:

- Implemento controlli avanzati per la password, come la verifica della lunghezza minima, la presenza di caratteri speciali, lettere maiuscole e minuscole e numeri. Questo aiuta gli utenti a creare password robuste, migliorando la resistenza del sistema agli attacchi di forza bruta e migliorando complessivamente la sicurezza del login.

Sicurezza con autenticazione a due fattori

L'autenticazione a due fattori (2FA) è un metodo di sicurezza informatica che aggiunge un secondo livello di verifica oltre alla tradizionale autenticazione tramite nome utente e password.

Autenticazione a Due Fattori con Google Authenticator:

- Ho integrato Google Authenticator utilizzando la libreria Sonata, che supporta l'autenticazione a due fattori. Questo sistema richiede oltre alla tradizionale combinazione di nome utente e password anche un codice generato dinamicamente dall'applicazione Google Authenticator installata sul dispositivo dell'utente.
- L'autenticazione a due fattori con Google Authenticator aggiunge un livello significativo di sicurezza. Anche se le credenziali di accesso (nome utente e password) vengono compromesse (attacchi di phishing o forza bruta), gli attaccanti non possono accedere all'account senza il codice generato dall'app Google Authenticator.

Protezione con Token CSRF:

- Ho implementato un meccanismo per generare e verificare un token CSRF (Cross-Site Request Forgery) per ogni richiesta POST.
- Il token CSRF garantisce che tutte le richieste POST siano autentiche e originate dall'utente legittimo, prevenendo così gli attacchi di tipo CSRF (tipo di attacco informatico che sfrutta l'identità autenticata di un utente in un'applicazione web per eseguire operazioni non autorizzate.) che potrebbero altrimenti manipolare o compromettere le operazioni dell'utente.

Gestione dei Secret e QR Code:

- Quando un utente accede alla propria area protetta, il sistema controlla se è già presente un "secret" nel database. Se non presente, viene generato un nuovo secret utilizzando GoogleAuthenticator e aggiornato nel database dell'utente. Successivamente, viene creato un URL QR Code utilizzando GoogleQrUrl per

visualizzare il QR Code che l'utente può scannerizzare con l'app Google Authenticator per configurare l'autenticazione a due fattori.

Download dei file

Il sistema di download degli articoli è progettato per garantire che solo gli utenti registrati possano accedere ai file. Questo controllo è implementato attraverso un sistema di autenticazione basato su sessioni, che richiede agli utenti di essere autenticati tramite login prima di poter procedere con il download.

Inoltre, per aumentare ulteriormente la sicurezza, la directory che contiene i file è protetta mediante l'utilizzo di un file .htaccess. Questo file è configurato per imporre restrizioni di accesso direttamente sul server web, assicurando che i file all'interno di quella directory non siano accessibili attraverso metodi non autorizzati.

Queste misure combinate non solo limitano l'accesso ai file solo agli utenti autorizzati, ma anche proteggono la directory da tentativi di accesso non autorizzato attraverso vulnerabilità comuni o tentativi di forzatura di URL.

Gestione delle Sessioni:

- La funzione session_start() è stata utilizzata per gestire le sessioni degli utenti. Verificando se $_SESSION['user_id'] è impostato, il codice garantisce che solo gli utenti autenticati possano accedere al download dei file. Se l'utente non è autenticato, viene reindirizzato alla pagina di login.

Sanitizzazione dei Dati:

- La funzione sanitize_filename($filename) è stata definita per sanificare il nome del file ricevuto tramite il parametro $_GET['file']. Questo approccio è cruciale per evitare attacchi di directory traversal e altri tipi di injection, garantendo che il nome del file sia sicuro e non possa essere usato per accedere a file al di fuori della directory consentita.

Verifica del Percorso del File:

- Prima di scaricare il file, viene verificato se il file esiste e se il percorso del file è all'interno della directory specificata (private/DOC). Questo controllo è fondamentale per impedire l'accesso non autorizzato a file al di fuori dell'area designata, utilizzando la funzione realpath() per confrontare i percorsi completi.

Headers di Sicurezza:

- Sono stati utilizzati diversi header HTTP (Content-Description, Content-Type, Content-Disposition, Expires, Cache-Control, Pragma, Content-Length) per indicare al browser come gestire il file scaricato. Questo non solo migliora l'esperienza utente ma previene anche attacchi di caching indesiderati e mantiene il controllo sull'integrità dei dati trasmessi.

Conclusioni

L’acronimo OWASP sta ad indicare un progetto open source nel campo della sicurezza delle applicazioni web-based che nasce nel 2001. L’iniziativa sin dall’anno della sua creazione ha sempre avuto come obiettivo principe quella di supportare sviluppatori,

tecnici e software house verso la creazione di applicativi sempre più sicuri, fornendo linee guida, best practice e pubblicazioni riconosciute a livello internazionale.

Tra le pubblicazioni più conosciute dalla fondazione senza dubbio c'è il progetto Top 10 OWASP. Si tratta di un elenco delle 10 vulnerabilità di sicurezza che colpiscono le applicazioni e le espongono al rischio di attacchi informatici.

I 10 rischi di sicurezza OWASP identificati nell'aggiornamento del 2021 sono i seguenti:

1. Broken Access Control
2. Crytpographic Failures
3. Injection
4. Insecure Design
5. Configurazione errata della sicurezza
6. Componenti vulnerabili e obsoleti
7. Errori di identificazione e autenticazione
8. Errori di integrità software e dati
9. Guasti di registrazione e monitoraggio della sicurezza
10. Falsificazione delle richieste lato server

Broken Access Control (Controllo di Accesso Non Funzionante):

- **Problema:** Questo rischio si verifica quando le restrizioni di accesso non sono implementate correttamente, consentendo agli utenti non autorizzati di accedere a risorse sensibili. Per proteggersi, è essenziale implementare controlli di accesso rigorosi, autenticazione e autorizzazione basata sui principi del principio del minimo privilegio.
- **Soluzione:** Nel mio sistema, ho implementato controlli di accesso utilizzando sessioni PHP (session_start() e $_SESSION['user_id']) per gestire l'autenticazione degli utenti e regolare l'accesso a risorse sensibili come il download di file. L'accesso diretto ai file tramite $_GET['file'] è sanificato per evitare accessi non autorizzati, mitigando così il rischio di Broken Access Control.

Cryptographic Failures (Fallimenti Crittografici):

- **Problema:** Questo rischio si verifica quando l'implementazione della crittografia è debole o non corretta, rendendo i dati sensibili vulnerabili agli attacchi. Per mitigare questo rischio, è fondamentale utilizzare algoritmi di crittografia robusti e aggiornati e gestire correttamente le chiavi crittografiche.
- **Soluzione:** Per garantire la sicurezza delle credenziali degli utenti, utilizzo la funzione PHP password_hash() per criptare le password prima di salvarle nel database. Questo approccio impiega algoritmi di hashing robusti per proteggere le credenziali dagli attacchi.

Injection:

- **Problema:** Le vulnerabilità di injection si verificano quando dati non fidati vengono inseriti in comandi o query per manipolare il comportamento del sistema. Per proteggersi, è importante utilizzare parametrizzazione delle query e validazione dei dati per prevenire attacchi di SQL injection, XSS (Cross-Site Scripting) e altri tipi di injection.

- **Soluzione:** Per prevenire le vulnerabilità di injection come SQL Injection e XSS, utilizzo funzioni PHP come sanitize_filename() e prepared statements con PDO per manipolare i dati del database. Queste tecniche assicurano che i dati non fidati inseriti nelle query non possano alterare il comportamento del sistema, mitigando efficacemente il rischio di Injection.

Progettazione Non Sicura:

- **Problema:** Questo rischio riguarda errori di progettazione che permettono a potenziali attaccanti di compromettere il sistema. Per mitigare questo rischio, è consigliabile adottare un approccio di sicurezza per design, integrando principi di sicurezza fin dall'inizio del processo di sviluppo del software.
- **Soluzione:** Implemento controlli di validazione lato client con JavaScript e PHP per i dati inseriti dagli utenti. Questo include l'autenticazionen a due fattori e l'uso di reCAPTCHA per proteggere contro tentativi automatizzati.

Configurazione Errata:

- **Problema:** Le configurazioni errate possono esporre involontariamente il sistema a rischi di sicurezza. Per proteggersi, è cruciale effettuare revisioni regolari delle configurazioni di sicurezza e utilizzare strumenti di analisi automatica per identificare e correggere configurazioni errate.
- **Soluzione:** Non ho trovato alcuna soluzione nel mio caso specifico.

Componenti Vulnerabili e Obsoleti:

- **Problema:** L'uso di componenti di terze parti vulnerabili o non aggiornati può esporre il sistema a rischi di sicurezza. Per mitigare questo rischio, è importante effettuare un inventario delle librerie e dei framework utilizzati e mantenere tutto aggiornato con le patch di sicurezza più recenti.
- **Soluzione:** Utilizzo di librerie sempre aggiornate.

Errori di Identificazione e Autenticazione:

- **Problema:** Errori nella gestione dell'identità e dell'autenticazione possono consentire agli attaccanti di accedere indebitamente al sistema. Per proteggersi, è fondamentale implementare procedure robuste di gestione delle credenziali, come autenticazione a più fattori e politiche di password sicure.
- **Soluzione:** L'uso di sessioni PHP per l'identificazione degli utenti ($_SESSION['user_id']) e la gestione delle credenziali di accesso con password robuste e autenticazione a due fattori (opzionale) aiutano a mitigare i rischi legati a errori di identificazione e autenticazione.

Fallimenti dell'Integrità Software e dei Dati:

- **Problema:** Questo rischio si riferisce alla manipolazione non autorizzata dei dati o del software, compromettendo l'integrità del sistema. Per mitigare questo rischio, è consigliabile utilizzare firme digitali, controlli di integrità dei file e tecniche di rilevamento delle modifiche non autorizzate.
- **Soluzione:** L'implementazione di controlli di validazione dei dati e la gestione corretta dei file scaricabili contribuiscono indirettamente a mantenere l'integrità dei dati e del software.

Guasti di Registrazione e Monitoraggio della Sicurezza:

- **Problema:** Una registrazione e un monitoraggio inadeguati possono ostacolare la capacità di rilevare e rispondere agli incidenti di sicurezza in modo tempestivo. Per proteggersi, è importante implementare logging dettagliato e monitoraggio continuo delle attività di sistema per identificare rapidamente comportamenti anomali.
- **Soluzione:** Non sono riuscito ad implementare un sistema di monitoraggio.

Falsificazione delle Richieste Lato Server:

- **Problema:** Questo rischio si verifica quando un attaccante induce il server a effettuare richieste indesiderate o dannose a risorse interne o esterne. Per mitigare questo rischio, è necessario validare e limitare le richieste in ingresso e implementare whitelist per le risorse accessibili.
- **Soluzione:** La gestione dei dati dell'utente è limitata e sanificata, riducendo i rischi di manipolazione delle richieste.

In conclusione, l'implementazione delle best practices e l'adozione di misure di sicurezza adeguate hanno contribuito significativamente a ridurre l'esposizione del mio sistema alle vulnerabilità elencate nella OWASP Top 10 2021, garantendo un ambiente più sicuro per gli utenti e proteggendo le risorse critiche dell'applicazione.