Corso di Sicurezza - relazione Homework

relazione a cura di Cristiano Romaldetti - matricola 2094675

Dimostrazione Vulnerabilità SQL Injection: Relazione di Progetto

Questa relazione presenta uno studio completo e un'implementazione pratica delle vulnerabilità di SQL injection attraverso un'applicazione web Node.js deliberatamente vulnerabile. Il progetto dimostra quattro classi distinte di attacchi SQL injection: attacchi basati su tautologie, bypass di autenticazione tramite commenti, esfiltrazione di dati tramite union-based e esecuzione di query concatenate (piggybacked). L'implementazione serve come framework educativo per comprendere le implicazioni di sicurezza di una validazione inadeguata dell'input e la compromissione della triade CIA (Confidenzialità, Integrità e Disponibilità) nelle applicazioni basate su database.

1. Introduzione

L'SQL injection rimane una delle vulnerabilità di sicurezza più diffuse e pericolose nelle applicazioni web, posizionandosi costantemente nella top 10 dei rischi di sicurezza OWASP. Questa vulnerabilità si verifica quando l'input dell'utente viene concatenato direttamente nelle query SQL senza una corretta sanitizzazione, permettendo agli aggressori di manipolare le operazioni del database oltre il loro scopo previsto.

L'obiettivo di questo progetto è creare un ambiente controllato che dimostri lo sfruttamento pratico delle vulnerabilità SQL injection, illustrando al contempo il loro impatto sulla sicurezza dei dati e sull'integrità del sistema.

2. Architettura del Progetto e Scelte Progettuali

2.1 Selezione dello Stack Tecnologico

Ambiente Runtime Node.js

La scelta di Node.js come ambiente runtime primario è motivata da diversi fattori:

- **Rilevanza Industriale**: Node.js ha guadagnato un'adozione significativa nello sviluppo web moderno, particolarmente per lo sviluppo di API e architetture di microservizi
- **Maturità dell'Ecosistema**: L'esteso ecosistema npm fornisce pacchetti immediatamente disponibili per la prototipazione rapida
- **Valore Educativo**: La natura dinamica di JavaScript rende più facile dimostrare come la mancanza di type safety possa contribuire alle vulnerabilità di sicurezza
- **Overhead Minimo del Framework**: A differenza di framework pesanti che spesso includono misure di sicurezza integrate, Node.js grezzo con Express.js fornisce una base pulita per introdurre intenzionalmente vulnerabilità

Sistema di Gestione Database SQLite

SQLite è stato selezionato come backend del database per le seguenti ragioni:

- **Semplicità**: Sistema di database a file singolo che elimina requisiti di configurazione complessi

- **Portabilità**: Database autocontenuto che consente facile distribuzione e deployment
- **Focalizzazione Educativa**: Riduce la complessità dell'infrastruttura, permettendo di concentrarsi sui concetti di sicurezza
- **Rilevanza Moderna**: SQLite è ampiamente utilizzato in applicazioni mobile, dispositivi IoT e sistemi embedded
- **Compatibilità SQL**: Supporta la sintassi SQL standard mantenendo un footprint leggero

Framework Web Express.js

Express.js fornisce un framework di applicazione web minimale che:

- **Abilita Sviluppo Rapido**: Creazione di API semplificata senza eccessive astrazioni
- **Mantiene Trasparenza**: Visibilità chiara del ciclo richiesta-risposta per scopi educativi
- **Supporta l'Introduzione di Vulnerabilità**: Manca di sanitizzazione dell'input integrata, permettendo difetti di sicurezza deliberati

2.2 Progettazione dello Schema del Database

Lo schema del database comprende quattro tabelle primarie progettate per simulare un'applicazione di e-commerce realistica:

```
-- Gestione utenti e autenticazione
CREATE TABLE users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT UNIQUE NOT NULL,
    password TEXT NOT NULL,
    email TEXT NOT NULL,
    role TEXT DEFAULT 'user',
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

-- Catalogo prodotti
CREATE TABLE products (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    price REAL NOT NULL,
    description TEXT,
    stock INTEGER DEFAULT 0
);

-- Registri delle transazioni
CREATE TABLE orders (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER,
    product_id INTEGER,
    quantity INTEGER,
    total_price REAL,
    order_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id),
    FOREIGN KEY (product_id) REFERENCES products(id)
);
```

```
-- Memorizzazione informazioni sensibili
CREATE TABLE sensitive_data (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER,
    credit_card TEXT,
    ssn TEXT,
    secret_notes TEXT,
    FOREIGN KEY (user_id) REFERENCES users(id)
);
```

Questo schema fornisce un contesto realistico per dimostrare come l'SQL injection possa compromettere diversi tipi di dati con livelli di sensibilità variabili.

3. Classi di Vulnerabilità Implementate

3.1 Attacchi Basati su Tautologie (Information Disclosure)

Endpoint: `GET /users?id=X`

Meccanismo della Vulnerabilità: L'applicazione concatena direttamente l'input dell'utente nella clausola WHERE di una query SQL:

```
const query = `SELECT id, username, email, role, created_at FROM users
WHERE id = ${userId}`;
```

Vettori di Attacco:

- **Tautologia di Base**: `?id=1 OR 1=1` bypassa la condizione WHERE, recuperando tutti i record utente
- **Scoperta Schema Database**: `?id=1 UNION SELECT name,sql,1,1,1 FROM sqlite_master WHERE type='table'` rivela la struttura del database
- **Informazioni di Versione**: `?id=1 UNION SELECT sqlite_version(),1,1,1,1` espone i dettagli della versione del database

Impatto sulla Sicurezza: Questa vulnerabilità consente l'accesso non autorizzato alle informazioni utente, esponendo potenzialmente indirizzi email, nomi utente e assegnazioni di ruoli che potrebbero facilitare ulteriori attacchi.

3.2 Bypass di Autenticazione Basato su Commenti

Endpoint: `POST /login`

Meccanismo della Vulnerabilità: Le credenziali utente vengono interpolate direttamente nella query di autenticazione:

```
const query = `SELECT id, username, email, role FROM users WHERE username =
'${username}' AND password = '${password}'`;
```

Vettori di Attacco:

- **Iniezione Commento SQL**: Username `admin'--` commenta la verifica della password
- **Tautologia nell'Autenticazione**: Username `admin' OR 1=1--` bypassa sia i controlli di username che password
- **Login Primo Utente**: Username `' OR '1'='1'--` autentica come il primo utente nel database

Impatto sulla Sicurezza: Completa elusione dei meccanismi di autenticazione, consentendo accesso non autorizzato a funzioni amministrative e account utente.

3.3 Esfiltrazione Dati Union-Based

Endpoint: `GET /search?query=X`

Meccanismo della Vulnerabilità: La funzionalità di ricerca incorpora direttamente l'input dell'utente nelle clausole LIKE:

```
const query = `SELECT id, name, description, price, stock FROM products
WHERE name LIKE '%${searchQuery}%'`;
```

Vettori di Attacco:

- **Estrazione Dati Sensibili**: `' UNION SELECT credit_card,ssn,secret_notes,1,1 FROM sensitive_data--` recupera informazioni confidenziali
- **Raccolta Credenziali Utente**: `' UNION SELECT username,password,email,role,1 FROM users--` espone dati di autenticazione
- **Analisi Struttura Database**: `' UNION SELECT name,sql,1,1,1 FROM sqlite_master WHERE type='table'--` rivela informazioni dello schema

Impatto sulla Sicurezza: Potenziale di violazione dati massiva, esponendo informazioni identificative personali, dati finanziari e credenziali di sistema.

3.4 Esecuzione di Query Concatenate (Manipolazione Dati)

Endpoint: `POST /update-profile`

Meccanismo della Vulnerabilità: La funzionalità di aggiornamento profilo consente multiple istruzioni SQL attraverso il metodo `db.exec()`:

```
const query = `UPDATE users SET email = '${bio}' WHERE username =
'${username}'`;
db.exec(query, callback);
```

Vettori di Attacco:

- **Escalation Privilegi**: `fake'; UPDATE users SET role='admin' WHERE username='john_doe'; --` promuove utenti regolari ad amministratori
- **Distruzione Dati**: `fake'; DROP TABLE products; --` elimina dati critici di business

- **Inserimento Dati Malevoli**: `fake'; INSERT INTO users (username,password,email,role) VALUES ('hacker','pwd','h@ck.er','admin'); --` crea account non autorizzati

Impatto sulla Sicurezza: Compromissione completa del sistema, consentendo manipolazione dati, escalation privilegi e interruzione del servizio.

4. Analisi dell'Impatto sulla Triade CIA

4.1 Compromissione della Confidenzialità

Scenario di Attacco: Estrazione union-based di dati sensibili **Impatto**: Divulgazione non autorizzata di:

- Numeri di carte di credito e informazioni finanziarie
- Codici fiscali e identificatori personali
- Note private e comunicazioni confidenziali
- Credenziali utente e dati di autenticazione

Dimostrazione:

```
GET /search?query=' UNION SELECT credit_card,ssn,secret_notes,1,1 FROM
sensitive_data--
```

4.2 Violazione dell'Integrità

Scenario di Attacco: Esecuzione di query concatenate per modifica dati **Impatto**: Alterazione non autorizzata di:

- Privilegi utente e livelli di accesso
- Credenziali account e informazioni di autenticazione
- Dati di business e registri delle transazioni
- Configurazione sistema e metadati

Dimostrazione:

```
POST /update-profile
bio: fake'; UPDATE users SET role='admin' WHERE username='john_doe'; --
```

4.3 Interruzione della Disponibilità

Scenario di Attacco: Distruzione tabelle database attraverso query concatenate **Impatto**: Indisponibilità del servizio attraverso:

- Eliminazione di tabelle critiche
- Corruzione del database
- Guasto delle funzionalità applicative
- Sospensione delle operazioni di business

Dimostrazione:

```
POST /update-profile
bio: fake'; DROP TABLE products; --
```

5. Risultati Sperimentali e Analisi

5.1 Tassi di Successo degli Attacchi

Tutti i vettori di attacco implementati hanno dimostrato tassi di successo del 100% nell'ambiente non protetto, confermando la natura critica dei fallimenti nella validazione dell'input.

5.2 Metriche di Esposizione Dati

- **Esposizione Dati Utente**: 5 account utente con informazioni credenziali complete
- **Informazioni Sensibili**: 5 record contenenti numeri di carte di credito, codici fiscali e note private
- **Dati di Business**: Catalogo prodotti completo e storico delle transazioni
- **Metadati di Sistema**: Schema database, strutture tabelle e informazioni di versione

5.3 Capacità di Escalation Privilegi

L'implementazione ha dimostrato con successo escalation di privilegi orizzontale e verticale:

- **Orizzontale**: Accesso ai dati e profili di altri utenti
- **Verticale**: Elevazione da privilegi utente ad amministratore
- **Livello Sistema**: Manipolazione schema database e distruzione tabelle

6. Valore Educativo e Risultati di Apprendimento

6.1 Consapevolezza della Sicurezza

Il progetto illustra efficacemente come decisioni implementative apparentemente minori possano portare a fallimenti catastrofici della sicurezza. La concatenazione diretta dell'input utente nelle query SQL, pur essendo sintatticamente semplice, crea gravi vulnerabilità.

6.2 Comprensione delle Strategie Difensive

Dimostrando attacchi di successo, il progetto evidenzia implicitamente misure difensive essenziali:

- **Query Parametrizzate**: Utilizzo di prepared statements con parametri vincolati
- **Validazione Input**: Implementazione di filtraggio input basato su whitelist
- **Principio del Privilegio Minimo**: Limitazione dei permessi dell'utente database
- **Codifica Output**: Sanitizzazione della presentazione dati per prevenire ulteriori sfruttamenti

6.3 Competenze di Valutazione del Rischio

Il progetto sviluppa competenze pratiche in:

- **Identificazione Vulnerabilità**: Riconoscimento di pattern di codifica pericolosi

- **Valutazione Impatto**: Comprensione delle conseguenze business dei fallimenti di sicurezza
- **Analisi Vettori di Attacco**: Comprensione di come input semplici possano compromettere interi sistemi

7. Conclusioni

Questo progetto di dimostrazione SQL injection illustra con successo l'importanza critica delle pratiche di codifica sicura nello sviluppo di applicazioni web. Attraverso l'implementazione di quattro classi distinte di vulnerabilità, il progetto dimostra come una validazione inadeguata dell'input possa compromettere tutti gli aspetti della triade CIA.

La scelta di Node.js e SQLite fornisce un contesto moderno e rilevante per comprendere queste vulnerabilità, riflettendo tecnologie comunemente utilizzate negli ambienti di sviluppo contemporanei. Le dimostrazioni di attacco complete servono come base pratica per comprendere sia i meccanismi tecnici dell'SQL injection sia le loro implicazioni di sicurezza più ampie.

Il valore educativo di questa implementazione si estende oltre la mera dimostrazione tecnica, fornendo intuizioni nelle pratiche di sviluppo sicuro, metodologie di valutazione del rischio e l'importanza critica dei principi di design security-first nello sviluppo software.

Romaldetti Cristiano - 2094675 - Sicurezza - Informatica LaSapienza - 29/09/2025