SQL Injection

Wojciech Trapkowski 193176, Łukasz Plust 186437

1 Anatomia ataku

SQL Injection to jeden z najczęstszych i najgroźniejszych ataków na aplikacje internetowe, oparty na manipulacji zapytaniami SQL, które są wysyłane do bazy danych. Atak ten polega na wstrzyknięciu do zapytania SQL złośliwego kodu, który zmienia jego pierwotne znaczenie. Ataki SQL są możliwe, gdy aplikacja internetowa nie waliduje lub nie odpowiednio sanitizuje danych wejściowych od użytkownika. W rezultacie, napastnik może uzyskać dostęp do danych, które normalnie są chronione, lub nawet przejąć kontrolę nad całą bazą danych.

1.1 Etapy ataku

1. Atakujący przeprowadza tzw. fuzzing, czyli wprowadza różne dane wejściowe do formularzy na stronie, testując, jak aplikacja na nie reaguje. Na przykład, może spróbować wpisać znak `'`, aby sprawdzić, czy aplikacja przetwarza go poprawnie, czy generuje błąd SQL. Przykładowe dane, które mogą być wprowadzane w polach formularza, to `1' OR '1'='1`, co w nieprawidłowo zabezpieczonej aplikacji może prowadzić do obejścia mechanizmu uwierzytelniania.

2. Gdy aplikacja nie stosuje odpowiednich zabezpieczeń, atakujący może wprowadzić zapytanie, które wprowadzi chaos w bazie danych. Przykładowo, zamiast standardowego zapytania SQL:

   ```
   SELECT * FROM users WHERE id = '1'
   ```

   atakujący może wprowadzić zapytanie, które uzyska wszystkie dane:

   ```
   SELECT * FROM users WHERE id = '1' OR '1'='1'
   ```

3. Jeśli atakujący zidentyfikuje lukę, może uzyskać dostęp do całej bazy danych. Może odczytywać, modyfikować, a nawet usuwać dane, takie jak hasła, informacje o użytkownikach czy dane osobowe. Bardziej zaawansowani atakujący mogą także uzyskać kontrolę nad systemem operacyjnym, jeśli baza danych ma zbyt szerokie uprawnienia.

2 Typy SQL Injection

2.1 Inband SQL Injection

Atakujący uzyskuje bezpośredni dostęp do danych z bazy, które są zwracane w odpowiedzi na zapytanie SQL. Przykładem jest wyświetlenie danych użytkownika na stronie internetowej po wykonaniu złośliwego zapytania.

2.2 Out-of-band SQL Injection

Atakujący nie otrzymuje odpowiedzi bezpośrednio z aplikacji, ale wysyła złośliwe zapytania SQL, które powodują, że baza danych kontaktuje się z zewnętrznym serwerem kontrolowanym przez atakującego. Przykład: wysyłanie złośliwych zapytań, które powodują, że baza danych wysyła dane do serwera HTTP kontrolowanego przez atakującego.

2.3 Blind SQL Injection

Atakujący nie otrzymuje bezpośredniego zwrotu danych z aplikacji, ale bazuje na różnych zachowaniach aplikacji, takich jak różnice w odpowiedziach, opóźnienia czasowe itp., aby wnioskować o wynikach zapytań SQL.

2.3.1 Typy Blind SQL Injection

1. **Oparte na warunkach logicznych:** Atakujący wprowadza warunki logiczne, które zmieniają wynik zapytania, np. `1=1` lub `1=2`, a następnie analizuje różnice w odpowiedzi aplikacji.

2. **Oparte na czasie:** Atakujący używa zapytań SQL, które celowo opóźniają odpowiedź serwera, np. poprzez użycie funkcji `SLEEP`. Jeśli aplikacja reaguje wolniej po wstrzyknięciu zapytania, atakujący wie, że jego warunek był prawdziwy.

3 Iniekcja kodu SQL a procedury składowane

Procedury składowane to funkcje zdefiniowane w bazie danych, które mogą przyjmować parametry i zwracać wyniki. Są to gotowe skrypty SQL, które mogą być wielokrotnie używane przez aplikacje. Procedury składowane mogą zwiększać wydajność i bezpieczeństwo aplikacji, jednak nie zawsze są one wolne od zagrożeń związanych z SQL Injection.

3.1 Zalety procedur składowanych w kontekście bezpieczeństwa

- **Bezpieczeństwo danych wejściowych:** Procedury składowane mogą być bardziej bezpieczne, gdyż parametry przekazywane do procedur są traktowane jako dane, a nie kod SQL.
- **Centralizacja logiki:** Wszystkie zapytania są przechowywane na serwerze bazy danych, co utrudnia atakującym manipulowanie zapytaniami bezpośrednio w kodzie aplikacji.

Jednak procedury składowane mogą być podatne na SQL Injection, jeśli korzystają z dynamicznego SQL – tj. gdy zapytania SQL są budowane w procedurze na podstawie danych wejściowych. Jeśli dane te nie są odpowiednio walidowane lub parametryzowane, procedury te są równie podatne na ataki jak zwykłe zapytania SQL.

3.2 Przykład podatności procedury składowanej

```
1 CREATE PROCEDURE GetUserDetails (@UserId NVARCHAR(50))
2 AS
3 BEGIN
4     EXEC('SELECT␣*␣FROM␣Users␣WHERE␣UserId␣=␣' +
          @UserId);
5 END
```

Wytłumaczenie: Jeśli `@UserId` jest pobierane bez walidacji od użytkownika, procedura ta jest podatna na SQL Injection. Atakujący może przekazać np. `' OR '1'='1` i uzyskać wszystkie dane użytkowników.

3.3 Ochrona przed SQL Injection w procedurach składowanych

Należy używać zapytania parametryzowanego zamiast dynamicznego SQL. Należy zawsze sprawdzać, czy dane są poprawne, zanim zostaną użyte w zapytaniu.

4 Techniki obrony przed SQL Injection

Aby skutecznie bronić się przed atakami SQL Injection, należy stosować kilka kluczowych zasad:

4.1 a) Zapytania parametryzowane

Najlepszą metodą zabezpieczenia się przed SQL Injection jest używanie zapytań parametryzowanych, w których dane wejściowe są traktowane jako parametry, a nie część kodu SQL.

```
import sqlite3

# Polaczenie z baza danych SQLite
conn = sqlite3.connect('example.db')
cursor = conn.cursor()

# Tworzenie tabeli
cursor.execute('CREATE TABLE IF NOT EXISTS users (id
    INTEGER PRIMARY KEY, username TEXT, password TEXT)'
    )

# Parametryzowane zapytanie
username = 'john_doe'
password = 'secure_password'
cursor.execute('SELECT * FROM users WHERE username = ?
    AND password = ?', (username, password))

# Zatwierdzenie i zamkniecie polaczenia
conn.commit()
conn.close()
```

4.2 b) Escapowanie danych wejściowych

Jeśli z jakiegoś powodu nie można użyć zapytań parametryzowanych, każda wartość pochodząca od użytkownika powinna być escapowana, tj. poddana procesowi, w którym specjalne znaki w danych wejściowych są traktowane jako zwykły tekst, a nie jako część kodu SQL lub innego języka programowania.

```
import sqlite3

# Escapowanie danych wejsciowych recznie
def escape_string(value):
    return value.replace("'", "''")  # Podwojny apostrof jest sposobem escapowania w SQL

conn = sqlite3.connect('example.db')
cursor = conn.cursor()

# Escapowane dane wejsciowe
username = "john_o'doe"  # Problem apostrofu
escaped_username = escape_string(username)

cursor.execute(f"SELECT * FROM users WHERE username = '{escaped_username}'")
result = cursor.fetchall()

conn.commit()
conn.close()
```

4.3 c) Walidacja i sanityzacja danych

Każde dane wejściowe powinny być walidowane pod kątem poprawności typu danych i formatu, zanim zostaną użyte w zapytaniu SQL. Powinna być również sprawdzana ich długość, zakres oraz zgodność z przewidywanym typem danych (np. liczba zamiast tekstu).

```
import sqlite3

def validate_user_input(user_input):
    if not user_input.isalnum():  # Sprawdzenie, czy dane wejsciowe zawieraja tylko litery i cyfry
        raise ValueError("Invalid input!")
```

```
6
7  # Uzycie funkcji walidacyjnej
8  username = 'john_doe'
9  validate_user_input(username)
10
11 conn = sqlite3.connect('example.db')
12 cursor = conn.cursor()
13
14 cursor.execute('SELECT * FROM users WHERE username = ?
       ', (username,))
15 result = cursor.fetchall()
16
17 conn.commit()
18 conn.close()
```

4.4 d) Ograniczenie uprawnień

Ważne jest, aby użytkownikom bazy danych przydzielano tylko te uprawnienia, które są absolutnie niezbędne do wykonywania ich zadań. Dzięki temu, nawet w przypadku udanego ataku SQL Injection, dostęp do krytycznych danych może być ograniczony.

4.5 e) Stosowanie ORM

Frameworki ORM (Object Relational Mapping) pomagają unikać bezpośrednich zapytań SQL, co minimalizuje ryzyko wstrzyknięcia kodu. ORM tłumaczy zapytania na SQL w sposób bezpieczny i bardziej kontrolowany.

```
1  from sqlalchemy import create_engine, Column, Integer,
       String
2  from sqlalchemy.ext.declarative import
       declarative_base
3  from sqlalchemy.orm import sessionmaker
4
5  # Silnik SQLite
6  engine = create_engine('sqlite:///example.db')
7  Base = declarative_base()
8
9  # Definicja modelu ORM
10 class User(Base):
11     __tablename__ = 'users'
12     id = Column(Integer, primary_key=True)
```

```
13      username = Column(String)
14      password = Column(String)
15
16  Base.metadata.create_all(engine)
17  Session = sessionmaker(bind=engine)
18  session = Session()
19
20  # Tworzenie nowego uzytkownika za pomoca ORM
21  new_user = User(username='john_doe', password='
        secure_password')
22  session.add(new_user)
23  session.commit()
24
25  # Pobieranie uzytkownika przez ORM
26  user = session.query(User).filter_by(username='
        john_doe').first()
27  print(user.username)
```

5 Automatyczna iniekcja kodu SQL

Automatyczna iniekcja kodu SQL polega na wykorzystaniu specjalistycznych narzędzi do automatycznego wykrywania luk i przeprowadzania ataków. Narzędzia te mogą automatycznie generować i wysyłać złośliwe zapytania SQL, zbierać informacje o bazie danych oraz wyciągać z niej dane.

5.1 Popularne narzędzia

- **sqlmap**: Automatyczne narzędzie do testowania aplikacji pod kątem podatności na SQL Injection, które umożliwia m.in. pobieranie bazy danych, uzyskiwanie haseł oraz wykonywanie poleceń systemowych.
- **Havij**: Graficzne narzędzie do automatycznych ataków SQL Injection. Pozwala na szybkie wykrycie i eksploatację luk.
- **BBQSQL**: Narzędzie open-source do blind SQL Injection, które automatyzuje testowanie aplikacji webowych pod kątem ślepych ataków SQL Injection.

6 Wskazówki dla programistów, administratorów i użytkowników

6.1 Wskazówki dla programistów

- Używanie zapytań parametryzowanych w każdej sytuacji, gdzie dane wejściowe są używane do budowania zapytań SQL.
- Walidacja danych wejściowych nie tylko po stronie serwera, ale także po stronie klienta. Dane powinny być zawsze sprawdzane pod kątem typu, zakresu i formatu.
- Unikanie dynamicznych zapytań SQL. W przypadku konieczności ich stosowania, należy upewnić się, że wszystkie dane są odpowiednio escapowane.
- Stosowanie ORM. Frameworki ORM automatycznie generują zapytania SQL, co redukuje ryzyko popełnienia błędów związanych z SQL Injection.

6.2 Wskazówki dla administratorów

- Minimalizacja uprawnień użytkowników bazy danych. Użytkownik bazy danych, z którego korzysta aplikacja, powinien mieć minimalne uprawnienia potrzebne do działania.
- Regularny monitoring logów bazy danych oraz aplikacji pod kątem podejrzanych zapytań SQL.
- Aktualizacja oprogramowania bazodanowego i aplikacji do najnowszych wersji, aby uniknąć znanych luk w zabezpieczeniach.
- Wdrożenie IDS/IPS (system wykrywania i zapobiegania włamaniom), który monitoruje ruch sieciowy pod kątem podejrzanych zapytań SQL.

6.3 Wskazówki dla użytkowników

- Stosowanie silnych haseł i unikalnych danych logowania w każdej aplikacji.
- Aktualizacja oprogramowania i przeglądarki do najnowszych wersji, aby zabezpieczyć się przed znanymi podatnościami.

- Unikanie podejrzanych stron internetowych i linków, które mogą być używane do phishingu i prób uzyskania dostępu do danych.