SQL Injections

https://github.com/christopherchristensen

Injection Basics

SQL-Injection-Angriff

- Einfügen (Injizieren) einer SQL-Abfrage (Query) über die Eingabedaten vom Client, um
  - sensible Daten aus der DB zu lesen
  - Daten in der DB zu modifizieren (CRUD)
  - Administrationsoperationen auf der DB auszuführen
  - Inhalt einer bestimmten Datei (auf DBMS) wiederherzustellen
  - Befehle an Betriebssystem auszugeben

Anwendungen von SQL Injections

- Anmeldeverfahren eines Chats erhacken, um Zugang zu schaffen

Wann werden SQL Injections ermöglicht?

- Wenn Entwickler dynamische DB-Abfragen erstellen,
  - die benutzerdefinierte Eingaben erhalten
  - und diese nicht filtern und maskieren (gegen böse Zeichen)
- z.B. bei Loginverfahren

Beispiel, um an Logindaten zu gelangen

- Folgender Query existiert auf Server

```
String accountBalanceQuery =
"SELECT accountNumber, balance FROM accounts WHERE account_owner_id = "
+ request.getParameter("user_id");

try {
```

```
        Statement statement = connection.createStatement();
        ResultSet rs = statement.executeQuery(accountBalanceQuery);

        while(rs.next()) {

            page.addTableRow(
              rs.getInt("accountNumber"),
              rs.getFloat("balance")
            );

        }

    } catch(SQLException e) { ... }
```

- Beabsichtigter Query

```
SELECT accountNumber, balance
FROM accounts
WHERE account_owner_id = 984
```

- Mit folgendem Query wird der Query immer `true`

```
SELECT accountNumber, balance
FROM accounts
WHERE account_owner_id = 0
OR [true statement]
```

- Somit gibt die Abfrage alle Kontonummern und Salden zurück von jedem Benutzer
- Wenn man für Passwort `'` eingibt kann man überprüfen, ob ein SQL-Syntax-Error geworfen wird oder nicht. Wenn nicht dann wird nicht gesäubert!

Massnahmen gegen SQL-Injections

- Prepared Statements
  - Sorgfältiges Überprüfen und Filtern von Eingaben und Parametern
- Stored Procedures
- White List Input Validation
- Web Application Firewall (WAF)

Advanced Injections

SQL Union

- Ergebnismenge von zwei oder mehr SELECT-Anweisungen kombinieren
- Beispiel

```
SELECT column-name(s) FROM table1
UNION
SELECT column_name(s) FROM table2
```

- Folgende Bedingungen müssen eingehalten werden,
  - Jede SELECT-Anweisungen innerhalb UNION muss gleiche Anzahl Spalten haben
  - Spalten müssen ähnliche Datentypen haben
  - Spalten in SELECT-Anweisung müssen gleiche Reihenfolge haben

SQL Union ALL

- UNION wählt per default nur eindeutige Werte aus
- Doppelte Werte zulassen mit,

```
SELECT City FROM Customers
UNION ALL
SELECT City FROM Suppliers
```

Union-based Angriffe

- Ermöglichen Informationen aus DB zu extrahieren
- Bedingungen
  - Beide Union Queries müssen übereinstimmen,
  - also muss Angreifer eine SELECT-Anweisung ähnlich zur ursprünglichen erstellen
  - Ein gültiger Tabellenname muss bekannt sein
  - Anzahl Spalten und Datentyp der ersten Query müssen bekannt sein
- Wenn Error-Reporting aktiv, wird Angriff erleichtert

Beispiel Union Anfriff über Suchmaske

1. Testen, ob Error-Reporting aktiv ist (mit `'`)
2. Mit Union-Anfrage Daten versuchen auszulesen

Lösung zu SQL-Injection Advanced

- Eingabe 1: `'`
- Eingabe 2:
  - `' UNION SELECT TABLE_NAME AS username FROM INFORMATION_SCHEMA.TABLES WHERE 1=1 or TABLE_NAME='`
  - `INFORMATION_SCHEMA.TABLES`
- Eingabe 3:
  - `' UNION SELECT pwd AS username FROM Users WHERE username='unionguy' =1 or surname='wrong`
- Eingabe 4:
  - `hash`

Mitigation von SQL-Injection (Advanced)

- Prepared Statements helfen
- Parametrisierte Abfragen zwingend Entwickler dazu,
  - SQL-Code zuerst zu definieren
  - Dann, jeden Parameter an Abfrage zu übergeben
- Ermöglicht Datenbank zwischen Code und Daten zu unterscheiden
- Weitere Möglichkeiten
  - Stored Procedures
  - White List Input Validation
  - Web Application Firewall (WAF) → second line of defense
- Passwörter nicht im Klartext in DB speichern