SQL-Injection

1. Was ist SQL-Injection?

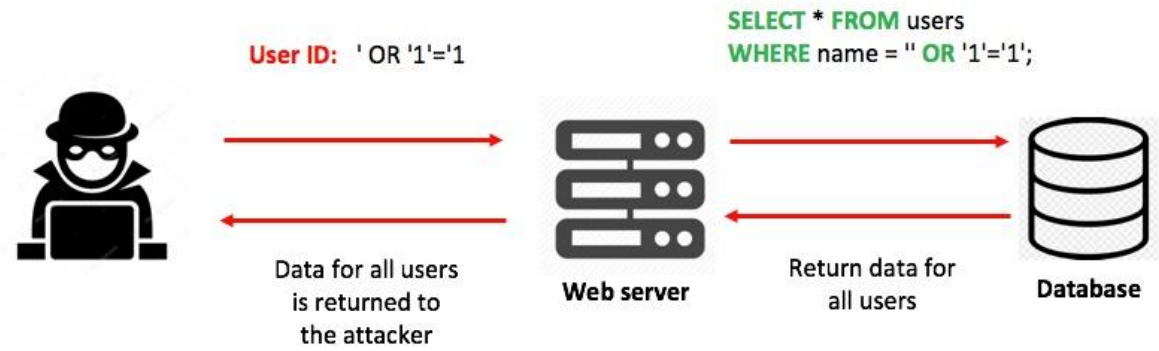
SQL Injection ist eine Sicherheitslücke im Web, die es einem Angreifer ermöglicht, die Abfragen zu stören oder zu ändern, die eine Anwendung an ihre Datenbank stellt. Im Allgemeinen kann ein Angreifer Daten anzeigen, die er normalerweise nicht abrufen kann. Dies kann Daten anderer Benutzer oder Anderes umfassen, auf die die Anwendung selbst zugreifen kann. In vielen Fällen kann ein Angreifer diese Daten ändern oder löschen, wodurch sich der Inhalt oder das Verhalten der Anwendung dauerhaft ändert. In einigen Situationen kann ein Angreifer einen SQL-Injection-Angriff eskalieren, und die ganze Datenbank löschen.

2. Welche Auswirkungen hat ein erfolgreicher Angriff?

Ein erfolgreicher SQL-Injection-Angriff kann zu unbefugtem Zugriff auf vertrauliche Daten wie Kennwörter, Kreditkartendaten oder persönliche Benutzerinformationen führen. Viele hochkarätige Datenschutzverletzungen in den letzten Jahren waren das Ergebnis von SQL-Injection-Angriffen, die zu Reputationsschäden (Reputation ist das Ansehen einer Person, sozialen Gruppe oder einer Organisation)und Bußgeldern führten. In einigen Fällen kann ein Angreifer eine dauerhafte Hintertür in die Systeme eines Unternehmens erhalten, was zu einem langfristigen Kompromiss führt, der über einen längeren Zeitraum unbemerkt bleiben kann

3. Szenarien

3.1. Szenario ohne Schutzmassnahmen mit einem Parameter



In diesem Beispiel sieht ihr rechts eine ‘SELECT’-Abfrage. Diese schaut, ob der Username übereinstimmt mit dem, der eingegeben wurde. Jedoch tricks nun der Angreifer dieses System aus, das nun überprüft wird, welcher User den Namen «» hat. Natürlich hat Niemand ein leeres Feld beim Namen, jedoch hängt er noch ein ‘OR’ hintendran. Dies bedeutet, bei allen Benutzer bei denen ‘1’=’1’ ist, werden angezeigt. Da ‘1’ immer gleich ‘1’ ist, bekommt der Angreifer alle Benutzer aus dieser Tabelle.

So könnte etwa der Code aussehen, die die Abfrage tätigt:

```php
public function sqlInjectionBackAction()
{
    $name = $_POST['name'];
    $db_instance = Database::getInstance();
    $conn = $db_instance->getConnection();
    $sql = "SELECT * FROM sql_injection_test WHERE name ='".$name."'";
    $stmt = $conn->prepare($sql);
    $stmt->execute();
    $res = $stmt->fetchAll(\PDO::FETCH_ASSOC);
    var_dump($res);
}
```

3.1. Szenario ohne Schutzmassnahmen mit zwei Parameter

Gleiches Szenario, einfach wird noch das Passwort abgefragt. Hier kann der Angreifer einfach nach dem Benutzernamen '--' einfügen, da dies denn Rest auskommentiert. Also zum Beispiel so: «' or 1=1 -- '».

```php
public function sqlInjectionTwoBackAction()
{
    $name = $_POST['name'];
    $pass = $_POST['pass'];
    $db_instance = Database::getInstance();
    $conn = $db_instance->getConnection();
    $sql = "SELECT * FROM sql_injection_test WHERE name ='".$name."' AND pass = '".$pass."'";
    $stmt = $conn->prepare($sql);
    $stmt->execute();
    $res = $stmt->fetchAll(\PDO::FETCH_ASSOC);
    var_dump($res);
}
```

4. Schutzmassnahmen

Die beste Methode, um solch ein Angriff abzuwehren, sind sogenannte «prepared Statements». Hier werden die Variablen so eingesetzt, das keine komischen Abfragen mehr kommen können. Hier seht ihr ein Beispiel:

```php
public function sqlInjectionNoBackAction()
{
    $name = $_POST['name'];
    $pass = $_POST['pass'];
    $db_instance = Database::getInstance();
    $conn = $db_instance->getConnection();
    $sql = "SELECT * FROM sql_injection_test WHERE name = :name AND pass = :pass";
    $stmt = $conn->prepare($sql);
    $stmt->bindValue(':name', $name);
    $stmt->bindValue(':pass', $pass);
    $stmt->execute();
    $res = $stmt->fetch(\PDO::FETCH_ASSOC);
    var_dump($res);
}
```

Prepared Statements werden verwendet, um die Sicherheit des Systems zu garantieren. Die Platzhalter ‘:name’ und ‘:pass’ werden erst im Datenbanksystem eingesetzt. Das bedeutet, dass das Datenbanksystem den String mit den Platzhaltern bekommt, und diese dann selbst die Variablen, die via ‘bindValue()’ übergeben werden, einsetzt. Anders als bei den vorherigen Beispielen, dort haben wir dies schon in PHP eingesetzt.