Sicherheitsüberlegungen

Beim Umgang mit Daten und Datenbanken müssen Sicherheitsüberlegungen von Beginn an bei der Anwendungsentwicklung berücksichtigt werden. Bei Applikationen, die Schnittstellen zum Internet haben, müssen solche Überlegungen eine noch zentralere Rolle spielen. Webapplikationen haben immer wieder schwerwiegende Sicherheitslücken, die dazu führen, dass sensitive Daten in falsche Hände oder die Öffentlichkeit gelangen.

Die OWASP listet in Ihrer "Top-Ten" die am häufigsten vorkommenden Fehler bei der Anwendungsentwicklung auf, auf Platz 1 befinden sich sogenannte "Injections" - diese Fehlerklasse betrachten wir im folgenden etwas genauer.

"Injection flaws"

> "Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker’s hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization."

Weiterführende Informationen

Eine Injection-Sicherheitslücke entsteht vereinfacht gesagt immer dann, wenn man (Benutzer-)Eingaben ohne weitere Validierung übernimmt und an weitere Befehle oder in weitere Verarbeitungsschritte weiterreicht. Das passiert sehr leicht, da man beim Programmieren für gewöhnlich nicht darüber nachdenkt, welche unsinnigen oder gar bösartigen Eingaben Benutzer möglicherweise machen, sondern meist darauf konzentriert ist, die erwarteten Eingaben effektiv weiterzuverarbeiten.

Grundregel der Webentwicklung: Vertraue keinem Datum, das dir ein Benutzer gibt. **Auf keinen Fall sollte man in einem produktiven System Benutzereingaben direkt in SQL Statements übernehmen.**[1)]

Eine **schlechte Idee** ist es also, das naheliegende zu tun:

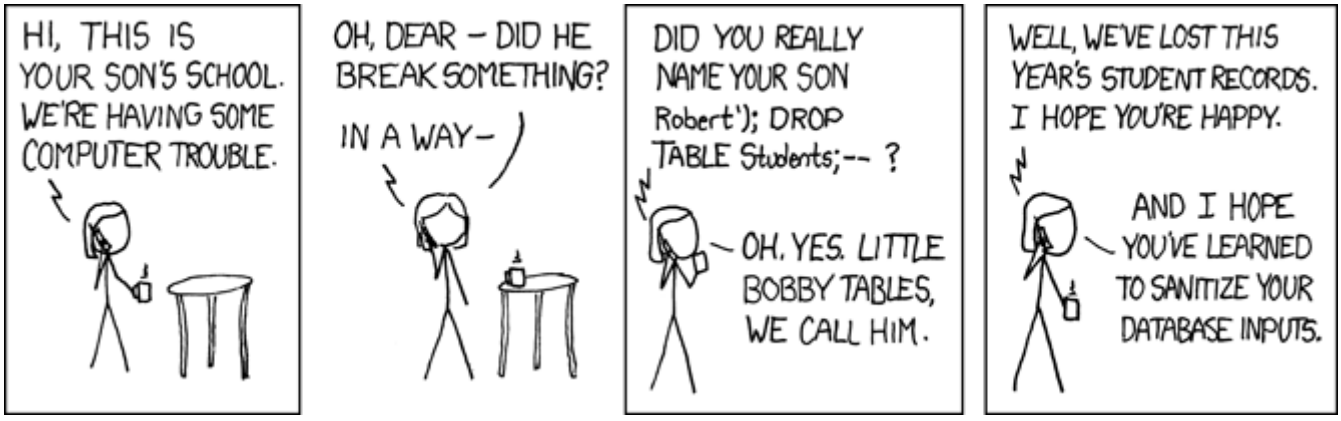
```php
// id wird in einem Formular vom Benutzer erfragt
if(isset($_POST['id'])) {
   $id = $_POST['id'];
} else {
   die(" Es muss eine Datensatz ID angegeben werden!");
}

echo "Datensatz mit der ID $id: <br>";
$sql = "SELECT * FROM schueler WHERE id = $id";
$rows = $pdo->query($sql) ;
foreach ($rows as $row) {
   echo $row['id'] . " " . $row['vorname']." ".$row['nachname']."<br />";
```

```
}
```

Dies funktioniert zwar, ist aber anfällig für sogenannte SQL Injections. Ein Angreifer kann über den POST-Parameter die SQL-Abfrage manipulieren und weiteren SQL-Code einschleusen. Im schlimmsten Fall werden dadurch sensible Daten ausgegeben, Tabelle verändert oder gar ganze Tabellen gelöscht. Gibt der Anwender nämlich ins Formularfeld beispielsweise folgendes ein: `1 OR id > 1`

Werden alle Datensätze ausgegeben, denn an die Datenbank wird die Abfrage `SELECT * FROM schueler WHERE id = OR id > 1` geschickt.



(Quelle: https://xkcd.com/327/, Lizenz Creative Commons Attribution-NonCommercial 2.5 License.

Prepared Statements

Um SQL-Injections zu unterbinden, sollte man prepared Statements verwenden. In dem Moment, in dem ihr Daten von Benutzern an die Datenbank übergebt, solltet ihr stets auf prepared Statements zurückgreifen.

```
// id wird in einem Formular vom Benutzer erfragt
if(isset($_POST['id'])) {
   $id = $_POST['id'];
} else {
   die(" Es muss eine Datensatz ID angegeben werden!");
}

echo "Datensatz mit der ID $id: <br>";

$statement = $pdo->prepare("SELECT * FROM schueler WHERE id = ?");
$statement->execute(array($id));

foreach ($row = $statement->fetch()) {
   echo $row['id'] . " " . $row['vorname']." ".$row['nachname']."<br />";
}
```

1)

https://www.ionos.de/digitalguide/server/sicherheit/sql-injection-grundlagen-und-schutzmassnahmen/

From:
https://www.info-bw.de/ -

Permanent link:
https://www.info-bw.de/faecher:informatik:oberstufe:datenbanken:projekt:dokuwiki_plugin:sicherheit:start?rev=1623243049

Last update: **09.06.2021 12:50**