

LU10c - SQLi Gegenmassnahmen

Glücklicherweise gibt gegen alle SQLi-Varianten entsprechende Gegenmassnahmen. Die hier vorliegende Liste ist daher nur ein Ausschnitt mit den prominentesten.

Escaping

Beim *escaping* werden Sonderzeichen nicht direkt in die Datenbank weitergeleitet. Die vom User eingegebenen Werte werden durch eine entsprechende Funktion/Methode, die es in vielen Programmiersprachen gibt vor der Weiterleitung gefiltert. Sonderzeichen, die den Angriffspunkt darstelle, werden von *gefährlichen* Steuerzeichen in *harmlose* Characters umgewandelt. Das „Escapen“ kann dabei auf der Client- und Serverseite geschehen.

Escaping – context cont.

- Different RDBMS have different ways of escaping data (it also depends on configuration)
- `addslashes()` works just like MySQL only „by chance“

RDBMS	PHP function	i've got quotes
PDO	<code>\$pdo->quote(\$val, \$type)</code>	n/a (it depends)
MySQL (mysql)	<code>mysql_real_escape_string</code>	<code>i\'ve got quotes</code>
MySQL (mysqli)	<code>mysqli_real_escape_string</code>	<code>i\'ve got quotes</code>
Oracle (oci8)	<code>n/d - str_replace()</code>	<code>i''ve got quotes</code>
SQLite	<code>sqlite_escape_string</code>	<code>i''ve got quotes</code>
MS SQL (mssql)	<code>n/d - str_replace()</code>	<code>i''ve got quotes</code>
PostgreSQL	<code>pg_escape_string()</code>	<code>i''ve got quotes</code>



28

Das Hauptproblem bei SQLi besteht darin, dass Werte ungeprüft in die Datenbank weitergegeben werden, wobei Platzhalter mit vom User zur Laufzeit eingegebenen Werten angereichert werden. Enthält eine solcher zur Laufzeit generierte SQL-Anweisung Injektion, wird diese ungeprüft und ungepuffert an die DB weitergeleitet.

Parametrisierte Werteübergabe

Die *parametrisierte Werteübergabe*, kurz *Parametrisierung* genannt, kann eine mögliche Gegenmassnahme gegen SQLi sein. Dabei werden die von User eingegebenen Werte in lokale

Parameter geschrieben. Diese Werte werden dann einen Schritt später durch Platzhalter in das SQL-Statement eingefügt.

Das nachfolgende JavaScript/MySQL-Beispiel soll diese Technik verdeutlichen:

```
// 1. Schritt: Zusammenbau des SQL-Statements mit Platzhaltern
const sql = `
  UPDATE Production.ProductInventory
  SET Quantity = @qty
  WHERE ProductID = @productId
`;
```

```
// 2. Schritt: Parametrisieren der Übergabewerte
const qty = 10;
const productId = 709;
```

```
// 3. Schritt: Zusammenbauen und Ausführen mit Laufzeitwerten
const sql = require('mssql');
async function updateInventory() {
  try {
    // Verbindung herstellen (Beispielconfig anpassen!)
    const pool = await sql.connect({
      user: 'username',
      password: 'password',
      server: 'localhost',
      database: 'AdventureWorks'
    });
    //
    const request = pool.request();
    request.input('qty', sql.Int, qty);
    request.input('productId', sql.Int, productId);
    //
    const result = await request.query(sql);
    console.log('Update erfolgreich:', result.rowsAffected);
  } catch (err) {
    console.error('Fehler beim Update:', err);
  }
}
```

updateInventory();



Volkan Demir

From:

<https://wiki.bzz.ch/> - **BZZ - Modulwiki**

Permanent link:

<https://wiki.bzz.ch/modul/m183/learningunits/lu10/03?rev=1758269151>

Last update: **2025/09/19 10:05**

