# L2\_2.3 Daten aus einer Datenbank auslesen und auswerten mit „Prepared-Statements“

Die gefundene Lösung zur Prüfung der Zugangsdaten eines Anwenders funktioniert fehlerfrei, hat aber einen großen Nachteil. Sie ist unsicher und kann sehr einfach angegriffen werden.



<?php

$name = $\_POST['name'];

$kennwort = $\_POST['kennwort'];

Hase

*//1. Verbindungsaufbau*

$pdo = new PDO("mysql:host=localhost;dbname=test;charset=utf8", "root", "");

*//2. Abfrage senden*

$sql = "SELECT loginname FROM login where loginname='$name' and kennwort='$kennwort'";

$stm = $pdo->query($sql);

*//3. Auswertung der DB-Antwort*

if ($row = $stm->fetchObject()) {

echo "Hallo " . $name . ", Sie sind nun angemeldet";

} else {

    echo "Hallo " . $name . ", die Zugangsdaten sind leider nicht korrekt.";

}

*//4. Verbindungsabbau*

$stm = null;

$pdo = null;

?>

Nehmen wir an, der Anwender füllt die Maske wie oben dargestellt aus, wählt also den Benutzername „Abele“ und das Kennwort „Hase“. Innerhalb der PHP-Seite wird nun eine SQL-Anweisung erzeugt, in welcher die Benutzereingabe integriert ist:

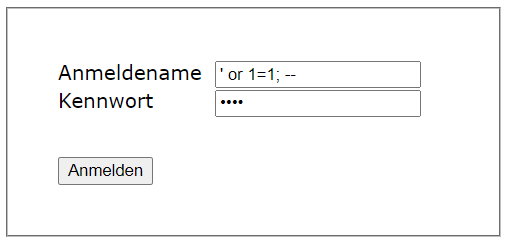
$sql = "SELECT loginname FROM login where loginname='$name' and kennwort='$kennwort'";

Setzt man die Eingaben ungefiltert ein, erhält man folgende Anweisung:

SELECT loginname FROM login where loginname='**Abele**' and kennwort='**Hase**'

Die SQL-Anweisung ist fehlerfrei und kann an das DBMS gesendet werden. Dies prüft, ob ein Datensatz vorhanden ist, bei dem in der Spalte „loginname“ der Wert „Abele“ hinterlegt ist und in der zugehörigen Spalte „kennwort“ der Wert „Hase“. Ist beides gegeben, liefert die Datenbank den Loginname“, ansonsten ist die Ergebnismenge leer.

Durch geschickte Wahl der Daten in der Eingabemaske, kann beispielsweise die Prüfung auf Korrektheit der Anmeldedaten umgangen werden. Nehmen wir an, der Anwender gibt folgende Daten in den Anmeldedialog ein:



egal

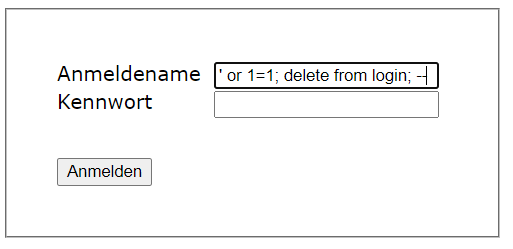
Unser PHP-Programm erzeugt daraus folgende SQL-Anweisung:

SELECT loginname FROM login where loginname='**' or 1=1; --**' and kennwort='**egal**'

Was ist passiert?

Das erste einfache Hochkommata schließt die zu vergleichende Zeichenkette hinter „loginname“. Diese ist nun leer. Zusätzlich wird die Anweisung durch ein logisches OR und einer zweiten Bedingung ergänzt, die immer(!) WAHR ist. Der Strichpunkt schließt die Anweisung ab, die beiden Minuszeichen kennzeichnen den Rest der Anweisung als Kommentar, der vom DBMS nicht weiter ausgewertet wird. Das Ergebnis der resultierenden SQL-Anweisung liefert somit alle Loginnamen aus der Datenbank.

Hat der Anwender mehr Kenntnisse über den Aufbau der Datenbank oder er rät den Namen der Tabelle mit den Zugangsdaten, kann er beispielsweise alle Datensätze löschen.



Diese Eingabe resultiert in der unten dargestellten SQL-Anweisung. In diesem Fall würden zuerst alle Loginnamen und anschließend alle Datensätze in der Tabelle „login“ entfernt. Die beiden Minuszeichen sorgen dafür, dass der Rest der Anweisung ignoriert wird.

SELECT loginname FROM login where loginname='**' or 1=1; delete from login; --**' and kennwort='**egal**'

## SQL-Injection

Für einen SQL-Injection-Angriff gibt ein Hacker spezielle SQL-Begriffe und Zeichen in ein Eingabefeld in einem Web-Formular ein, um die Anwendung dazu zu bringen, andere Befehle als normalerweise an die Datenbank zu senden.

Basierende auf die Eingaben des Besuchers der Seite sendet die Web-Anwendung SQL-Anweisungen an das DBMS. Bei einem SQL-Injection-Angriff schleust ein in SQL-Syntax sehr versierter Hacker über geschickte Wahl der Eingabedaten innerhalb der bereitgestellten Formulare Einträge mit dem Ziel ein, einen direkteren und weitreichenderen Zugriff auf das DBMS zu erhalten, als es die Web-Anwendung vorsieht. In vielen Fällen wird mit solchen Angriffen versucht, wertvolle Informationen wie Zugangsdaten oder vertrauliche Finanz- und Unternehmensdaten abzurufen. Wie wir im zweiten Beispiel gesehen haben, können Daten auch manipuliert oder gar gelöscht werden.

## Prepared Statements

Einer SQL-Injection kann durch eine Analyse und Filterung der Formulareingaben entgegengewirkt werden. Ein solches Vorhaben kann jedoch sehr aufwendig sein. Um SQL-Injections zu verhindern, biete sich der Einsatz von „prepared statements“ an. Dabei handelt es sich um eine spezielle Art der Erstellung einer SQL-Anweisung. Die Anweisungen werden unter Verwendung von Platzhaltern vordefiniert und dem DBMS bereitgestellt. In einem zweiten Schritt werden dann die konkreten Werte für die Platzhalter an das DBMS gesendet. Das DBMS prüft die Gültigkeit der Parameter bevor diese verarbeitet werden. Die Vorgehensweise ist dabei sehr ähnlich der bisher bekannten.

1. Die SQL-Anweisung wird wie bisher formuliert, wobei die konkreten Werte durch den Platzhalter „?“ ersetzte werden. Es handelt sich also um eine Art Muster der Abfrage. Vereinfach gesagt, wird dieses Muster durch Aufruf der Methode „prepare(...)“ an das DBMS gesendet.

$sql = "SELECT loginname FROM login where loginname=? and kennwort=?";

$stm = $pdo->prepare($sql);

Beachten Sie, dass die Fragezeichen frei stehen und auch im Falle, dass sie eine Zeichenkette repräsentieren keine umschließende Hochkommata haben!

1. In einem zweiten Schritt werden nun die konkreten Werte für die Platzhalter übermittelt. In unserem Fall handelt es sich dabei um den konkreten Anmeldename (Variable $name) und das vom Anwender eingegebene Kennwort (Variable $kennwort). Da es sich um mehr als ein Platzhalter handeln kann, eigenen sich ein Array als Datenstruktur, welches durch Aufruf der Methode „execute(…)“ an das DBMS gesendet wird.

$stm->execute( array($name, $kennwort) );

Im weiteren Programmverlauf sind keine weiteren Änderungen vorzunehmen, so dass das Ergebnis sich wie folgt präsentiert:

<?php

$name = $\_POST['name'];

$kennwort = $\_POST['kennwort'];

$pdo = new PDO("mysql:host=localhost;dbname=test;charset=utf8", "root", "");

$sql = "SELECT loginname FROM login where loginname=? and kennwort=?";

$stm = $pdo->prepare($sql);

$stm->execute(array($name, $kennwort));

if ($row = $stm->fetchObject()) {

    echo "Hallo " . $row->loginname . ", Sie sind nun angemeldet";

} else {

    echo "Hallo " . $name . ", die Zugangsdaten sind leider nicht korrekt.";

}

$stm = null;

$pdo = null;

?>

Soll ein Statement mit unterschiedlichen Parametern mehrere Male (z. B. innerhalb einer Schleife) auf dem Datenbanksystem ausgeführt werden, können Prepared Statements auch einen deutlichen Geschwindigkeitsvorteil bringen, da das Statement schon vorübersetzt (prepared) im Datenbanksystem vorliegt und nur noch mit den neuen Parametern ausgeführt werden muss.

**Für die Profis:**

Zur Erhöhung der Lesbarkeit des Programmcodes können anstelle des Platzhalters „?“ auch eigene Platzhalter formuliert werden. Diese beginnen immer mit einem Doppelpunkt. Das Beispiel von oben kann damit wie folgt realisiert werden:

$sql = "SELECT loginname FROM login where loginname=**:ln** and kennwort=**:kw**";

$stm = $pdo->prepare($sql);

$stm->execute( array(':**ln**'=>$name, ':**kw**'=>$kennwort) );

Man spricht hier auch von assoziativen Arrays. Diese zeichnen sich dadurch aus, dass jedes Array-Element einen Namen und einen Wert besitzt. Die Erstellung eines solchen Arrays hat dann die folgende Struktur:

array( 'name1'=>wert1, 'name2'=>wert2, 'name3'=>wert3,… )

**Ergänzung:**

Möchte man nicht mit (assoziativen) Arrays arbeiten, können die Parameter auch über die Methode „bindParam(…)“ an die Platzhalter gebunden werden.

$sql = "SELECT loginname FROM login where loginname=? and kennwort=**?**";

$stm = $pdo->prepare($sql);

$stm->bindParam(1, $name);

$stm->bindParam(2, $kennwort);

$stm->execute();

Im Beispiel wird der erste Platzhalter (?) durch den Wert ersetzt, der in der Variablen $name steht, der zweite Parameter durch den Wert der Variablen $kennwort.

Wie wir bereits wissen, können zur Erhöhung der Lesbarkeit des Programmcodes anstelle des Platzhalters „?“ auch eigene Platzhalter formuliert werden. Man spricht auch von benannten Platzhaltern. Diese beginnen immer mit einem Doppelpunkt. Das Beispiel könnte somit auch wie folgt aussehen:

$sql = "SELECT loginname FROM login where loginname=**:ln** and kennwort=**:kw**";

$stm = $pdo->prepare($sql);

$stm->bindParam(':ln', $name);

$stm->bindParam(':kw', $kennwort);

$stm->execute();