PHP: Datenbankzugriff mit MySQLi, Authentifizierung

- Die Sicherheitslücke der Serverausgaben am Browser schließen können
- Die Möglichkeiten von MySQLi zum Zugriff auf MySQL im Hinblick auf Sicherheit anwenden können
- Datenmanipulations- und Datenabfrageanweisungen über MySQLi absetzen können
- Daten unterschiedlichster Typen über MySQLi verarbeiten können
- Den Begriff SQL-Injection einordnen und entsprechende Sicherheitsmaßnahmen ergreifen können
- Den richtigen Umgang mit Passwörtern beim Abspeichern in der Datenbank beherrschen
- Eine einfache aber komfortable Benutzerauthentifizierung realisieren können
- Sicherheitsproblem Cross-Side-Request-Forgery verstehen

SICHERHEITSPROBLEM

Während der Entwicklung (*Development*) der Anwendung sind Fehlermeldungen äußerst hilfreich aber im laufenden Betrieb (*Production*) gibt jede Fehlermeldung für einen Hacker wichtige Hinweise preis:

- Datenbank- und Tabellenname
- Interne Struktur der Web-Anwendung

Lösung (in `php.ini` einstellbar)

`display_errors = Off` (*Production*) / `On` (*Development*)
Errors, Warnings und Notices werden nicht angezeigt

`log_errors = On` (*Production*) / `Off` (*Development*)
Errors, Warnings und Notices werden in die Log-Datei `xampp/apache/log/error.log` geschrieben[1]

HINWEIS: Mit `trigger_error()` können Fehlermeldungen geworfen werden (siehe hinten)

MySQLi

Erweiterung von PHP zum Zugriff auf MySQL-Datenbanken

- kann prozedural und objektorientiert verwendet werden
- Mit *Prepared Statements* können SQL-Injection-Angriffe verhindert werden (siehe hinten)

[1] Diese können auch automatisch über E-Mail an Administrator geschickt werden

Daten einfügen, ändern und löschen mit Prepared Statement

$con

- **Verbindungsaufbau**

```
$con = new MySQLi(
  "localhost", "root", "masterkey", "users");
```

$sql

- **SQL-Befehl erstellen**

```
$sql = "INSERT INTO users(uusername, upassword,
  umale, ubirthdate, urating, uimagetype,
  uimagesize, uimage) VALUES (?,?,?,?,?,?,?,?);";
```

$stmt

- **Prepared Statement erstellen**

```
$stmt = $con->prepare($sql);
```

- **Parameter binden**

```
$stmt->bind_param("ssisdsib", $username, $password,
  $male, $birthDate, $rating, $imageType,
  $imageSize, $null);
```

- **Parameter mit Werten füllen**

```
$username = $user->getUsername();
...
```

- **Prepared Statement ausführen**

```
$stmt->execute();
```

- Bei Erfolg

```
$stmt->affected_rows und $stmt->insert_id
```

- `$stmt` und `$con` schließen

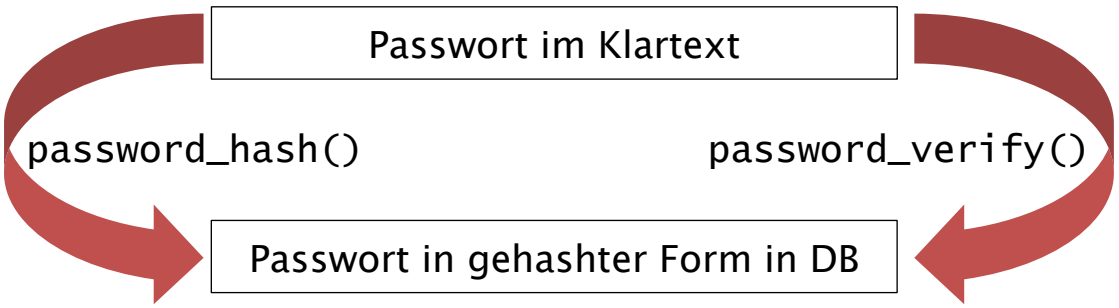
```php
<?php
$con = new MySQLi("localhost", "root", "masterkey", "users");
if ($con->connect_errno) {
  // Meldung wird bereits ausgegeben bzw. in den Log geschrieben
  // Skript wird nicht abgebrochen
} else {
  $sql = "
    INSERT INTO users(uusername, upassword, umale, ubirthdate,
      urating, uimagetype, uimagesize, uimage)
      VALUES (?, ?, ?, ?, ?, ?, ?, ?);
  ";
  $stmt = $con->prepare($sql);
  if ($con->errno) {
    // Meldung muss händisch geworfen werden
    trigger_error($con->error, E_USER_WARNING);
  } else {
    $stmt->bind_param("ssisdsib", $username, $password, $male,
      $birthDate, $rating, $imageType, $imageSize, $null);
    $username = $user->getUsername();
    $password = password_hash(
      $user->getPassword(), PASSWORD_DEFAULT);
    $male = $user->getMale();
    $birthDate = $user->getBirthDate();
    $rating = $user->getRating();
    $imageType = $user->getImageType();
    $imageSize = $user->getImageSize();
    $null = null;
    $stmt->send_long_data(7, $user->getImage());
    $stmt->execute();
    if ($con->errno) {
      if ($con->errno == 1062)
        $user->setError(
          "username", "Benutzername bereits vergeben");
      else
        trigger_error($con->error, E_USER_WARNING);
    } else {
      echo "Anzahl eingetragene Datensätze ".$stmt->affected_rows;
      echo "Primärschlüssel neuer Benutzer ".$stmt->insert_id;
    }
    $stmt->close();
  }
  $con->close();
}
```

- `prepare()` erzeugt *Prepared Statement* durch welches *SQL-Injection Angriffe* verhindert werden (siehe hinten)
- Bei `prepare()` kontrolliert DB-Server Anweisung auf Syntax-Fehler und setzt `$con->errno` und `$con->error`
- Bei `E_USER_WARNING` wird Skript normal fortgesetzt, bei `E_USER_ERROR` würde es sofort abgebrochen
- Bei `bind_param()` müssen Variablen übergeben werden, Funktionsaufrufe sind nicht erlaubt
- Mit `password_hash()` verschlüsselt PHP durch Standard-Verschlüsselungsalgorithmus das Passwort welches in gehashter Form in Datenbank abgespeichert wird. Das Datenfeld muss mind. 60 Zeichen lang (hängt vom Verschlüsselungsalgorithmus ab).
  `PASSWORD_DEFAULT` gibt an, dass die momentan stärkste Hash-Einwegverschlüsselung verwendet werden soll

  `password_verify()` testet Passwort auf Korrektheit
- Datum muss als String im Format `"Y-m-d"` übergeben werden[2]
- Verbindungs- (`$con`) und Statement-Objekt (`$stmt`) müssen nach Gebrauch zerstört werden
- **WICHTIG**: `$stmt->affected_rows` und `$stmt->insert_id`

SICHERHEITSPROBLEM

Passwörter sollen in Datenbank in gehashter Form abgelegt werden:



[2] Internationales Datumsformat gemäß ISO 8601

Daten abrufen mit Prepared Statement

```
$con = new MySQLi("localhost", "root", "masterkey", "users");
if (!$con->connect_errno) {
  $sql = "
    SELECT uusername, upassword, umale, ubirthdate, urating,
      uimagetype, uimagesize, uimage
    FROM users
    WHERE uusername = ?;
  ";
  $stmt = $con->prepare($sql);
  if ($con->errno) {
    trigger_error($con->error, E_USER_WARNING);
  } else {
    $stmt->bind_param("s", $username);
    $stmt->execute();
    $stmt->store_result();
    $stmt->bind_result($username, $password, $male, $birthDate,
      $rating, $imageType, $imageSize, $image);
    if ($stmt->fetch()) {
      $user = new ValidableUser($username, null, null, $male,
        $birthDate, $rating, $imageType, $imageSize, $image);
      $ret = $user;
    }
    $stmt->close();
  }
  $con->close();
}
```

- Sollte Ergebnismenge aus mehreren Datensätzen bestehen dann `while($stmt->fetch()) { … }`
- Datum wird als String im Format `"Y-m-d"` geliefert
- `store_result()` holt gesamtes Ergebnis vom Server. Aufruf ist wichtig, wenn geschachtelte Prepared-Statements gegenseitig auf Ergebnisse zugreifen
- Liegen die Datenbankinhalte nicht im Zeichensatz UTF-8 vor, so müssen die Zeichenketten mit `utf8_encode()` konvertiert werden. Wurden die Zeichenketten über PHP in die Datenbank geschrieben, so ist beim Schreiben sowie beim Lesen keine Konvertierung notwendig

SICHERHEITSPROBLEM SQL-Injection

FRAGE: Wieso anstelle von *Statement* (`$con->query()`) *Prepared Statement* (`$con->prepare()`) verwenden?

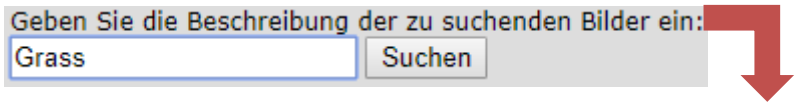
Beispiel Bildersuche

```
$sql = "
  SELECT iid, iowner, itype, isize, iimage
    FROM images
    WHERE iowner=\"Sepp\" AND idescription=\"$_GET[seachString]\";
  ";
$result = $con->query($sql);
while ($row = $result->fetch_assoc()) {
  …
```

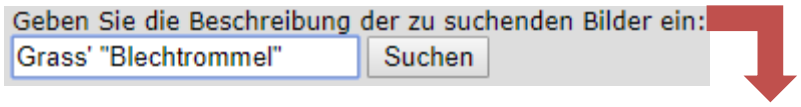
Statement

```
$sql = "
  SELECT iid, iowner, itype, isize, iimage
    FROM images
    WHERE iowner = \"Sepp\" AND idescription = ?;
  ";
$stmt = $con->prepare($sql);
$stmt->bind_param("s", $searchString);
$searchString = $_GET["searchString"];
$stmt->execute();
…
```

Prepared Statement



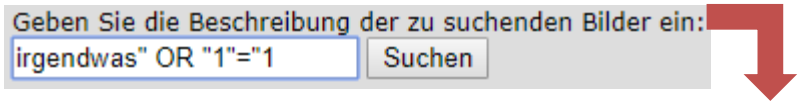
`SELECT … AND idescription="Grass";` ☺ *Statement*



`… AND idescription="Grass' "Blechtrommel"";` ☹ *Statement*

1. Grund wieso Prepared Statement:

Prepared Statement maskiert SQL-Sonderzeichen %, _, ", und ' ☺



`iowner = "Sepp" AND idescription="irgendwas" OR "1"="1";`
liefert alle (!) Bilder ☹ *Statement*

2. Grund wieso Prepared Statement:

Prepared Statement lässt bereits beim Aufruf von `prepare()` DB-Server SQL-Anweisung validieren und tauscht bei Parameter nur einzelne Werte (`int`, `double`, `string`, `binary`) aus

Aufgabe

Auf jedem MySQL-Datenbankserver gibt es die Datenbank `information_schema` welche Meta-Informationen über alle am Server verwalteten Datenbanken enthält. Diese Datenbank enthält die Tabelle `columns` welche Informationen über alle Datenfelder aller verwalteten Datenbanken enthält. Die Tabelle besteht u.a. aus folgenden Spalten: `table_schema` (Datenbankname), `table_name`, `column_name` und `data_type`.

Was würden Sie ins vorige Formularfeld eingeben um Informationen über alle Spalten aller Tabellen der Datenbank `users` zu erhalten (**TIPPS**: `UNION`. Um im `SELECT`-Befehl auf `columns` in `information_schema` zuzugreifen schreiben Sie …`FROM information_schema.columns`…)

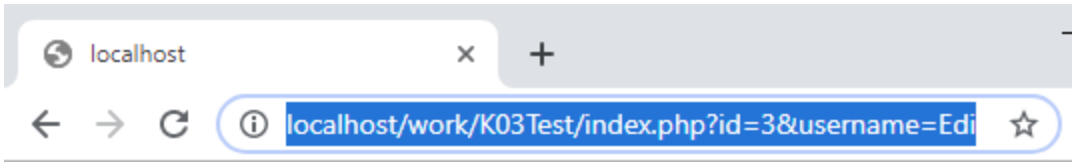
SICHERHEITSPROBLEM

Benutzer werden in einer Liste nur bestimmte Datensätze angezeigt, weil er nur diese bearbeiten darf:

Benutzername	Geschlecht	Geburtsdatum	Bewertung		
Elas	Weiblich	15.01.1999	3,25	Bearbeiten	Löschen
Resi	Weiblich	01.07.2000	4,75	Bearbeiten	Löschen
Sepp	Männlich	09.10.2019	3,00	Bearbeiten	Löschen

```
<a href="index.php?id=3&username=Elsa">Bearbeiten</a>
<a href="index.php?id=3&username=Resi">Bearbeiten</a>
<a href="index.php?id=3&username=Sepp">Bearbeiten</a>
```

Benutzer muss daran gehindert werden, durch Eingabe von



auf andere Benutzer zuzugreifen

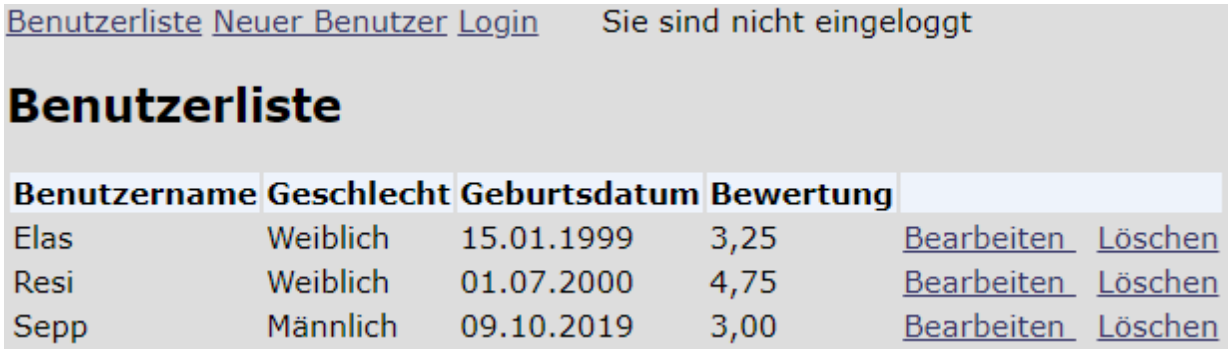
Lösung

Beim Holen eines Datensatzes aus der Datenbank muss kontrolliert werden, ob Benutzer tatsächlich dazu berechtigt ist

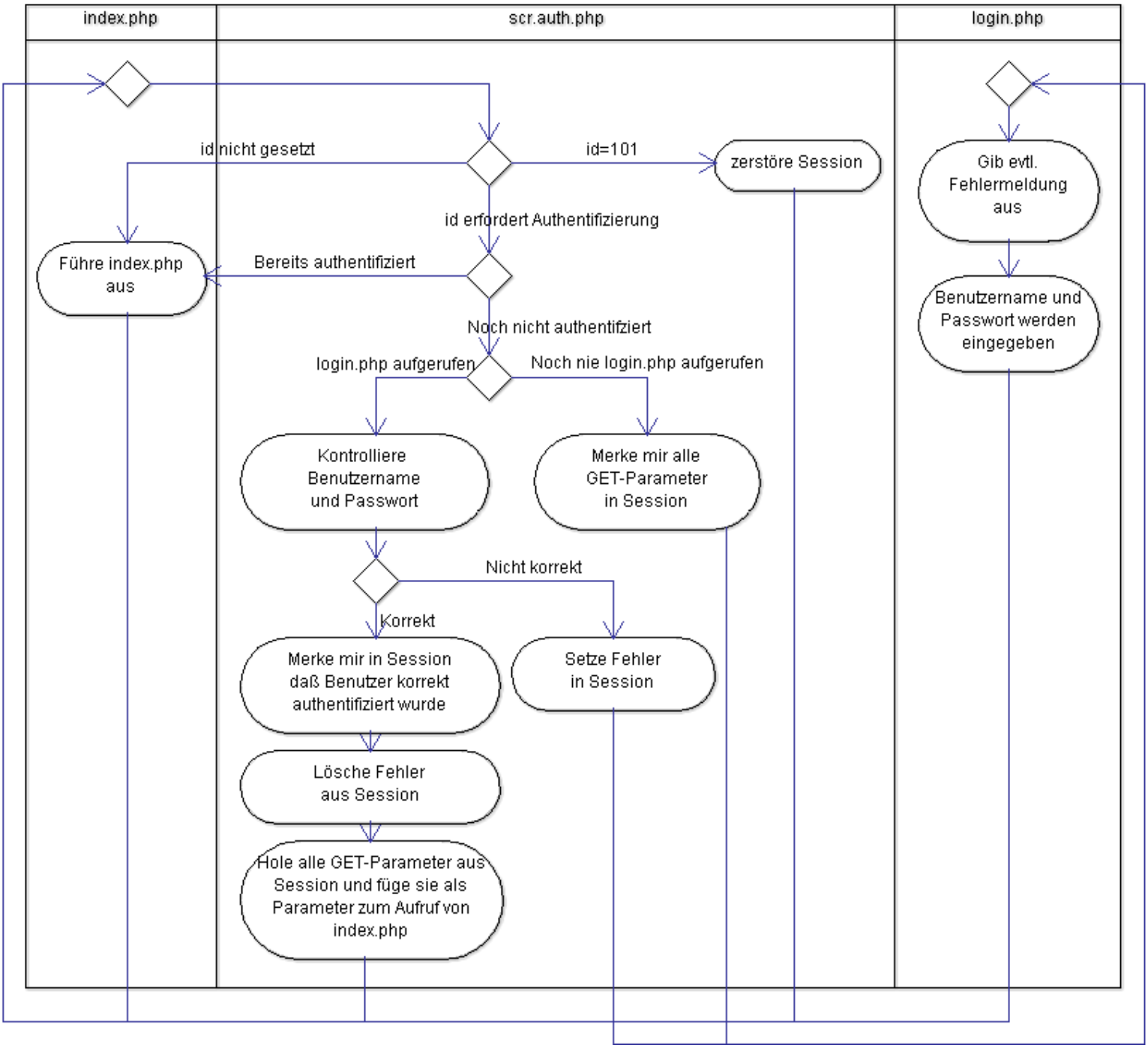
Einfache Benutzerauthentifizierung

HINWEIS: PHP-Frameworks wie *Zends* enthalten Komponenten für die sichere und professionelle Authentifizierung, Benutzerverwaltung, Datenbankanbindung u.v.m.

Anforderungen



- Bestimmte Ids erfordern Authentifizierung des Benutzers (*Neuer Benutzer* Id=2, *Bearbeiten* Id=3, *Löschen* Id=4)
- Andere Ids (insbesondere der Aufruf von `index.php` ohne Id) erfordern keine Authentifizierung (*Benutzerliste* Id=1)
- Wird beispielsweise bei noch nicht erfolgter Authentifizierung auf *Bearbeiten* geklickt, muss automatisch Login-Formular aufgeworfen werden, und bei erfolgreicher Authentifizierung automatisch das Bearbeiten-Formular zur Verfügung gestellt werden
- Hat sich Benutzer einmal korrekt authentifiziert, wird keine weitere Authentifizierung verlangt
- Benutzer soll sich *ein- und ausloggen* können (*Login* Id=100, *Logout* Id=101)



- `login.php` im Wurzelordner des Projekts, `scr.auth.php` im `scripts`-Ordner
- Ist `$_SESSION["loginUsername"]` gesetzt bedeutet dies, dass Benutzer korrekt authentifiziert wurde
- Ist `$_SESSION["requested_id"]` gesetzt so bedeutet dies, dass Login-Vorgang eingeleitet wurde

```php
<?php
const IDS_LOGIN_NOT_NECESSARY = ["1"];

if (isset($_GET["id"])) {
  if ($_GET["id"] == 101) {
    // Logout
    session_destroy();
    header("Location:index.php");
    // Bricht Abarbeitung des Skripts ab
    exit();

  } else {
    if (array_search($_GET["id"], IDS_LOGIN_NOT_NECESSARY, true)
       === false && !isset($_SESSION["loginUsername"])) {
      // Benutzer muss sich für gewünschte Operation auth.
      // und hat sich noch nicht authentifiziert

      if (! isset($_SESSION["requested_id"])) {
        // Für Benutzer wurde noch nie login.php aufgerufen
        // Merke welche id und Parameter Benutzer angefordert hat
        foreach ($_GET as $key => $value)
          $_SESSION["requested_" . $key] = $value;
        header("Location:login.php");
      } else {

        // Für Benutzer wurde login.php aufgerufen
        // Kontrolliere ob Benutzername und Passwort passen
        $username = filter_input(INPUT_POST, "loginUsername",
          FILTER_SANITIZE_STRING);
        $password = filter_input(INPUT_POST, "loginPassword",
          FILTER_SANITIZE_STRING);
        if (!UserList::authUser($username, $password)) {
          $_SESSION["loginError"] = true;
          header('Location:login.php');

        } else {
          // Benutzername und Passwort passen
          $_SESSION["loginUsername"] = $username;
          unset($_SESSION["loginError"]);
          if ($_SESSION["requested_id"] == "100")
            unset($_SESSION["requested_id"]);
          $params = "";
          foreach ($_SESSION as $key => $value)
            if (strpos($key, "requested_") === 0) {
              if (!empty($params))
                $params .= "&";
              else
                $params = "?";
              $params .= substr($key, 10) . "=" . $value;
              unset($_SESSION[$key]);
            }
          header("Location:index.php" . $params);
        }
      }
      exit();
    }
  }
}
```

SICHERHEITSPROBLEM: Cross-Side-Request-Forgery (XSRF)

Server der die Authentifizierung mit Benutzername und Passwort durchführt muss wissen, dass Authentifizierungsformular von dem Eingabedaten übermittelt wurden, von ihm generiert wurde denn sonst könnten Angreifer präparierte Login-Attacken auch über E-Mail-Links durchführen.

Lösung

Der Server schickt an den Browser mit dem Formular einen ständig wechselnden Code der vom Browser mit Benutzername und Passwort wieder zurück an Server geschickt wird. Server kontrolliert diesen Code und erkennt, dass dieser von ihm stammt und kann Authentifizierung durchführen. Wäre Code nicht vorhanden oder stimmt er nicht mit dem versendeten Code überein wird Authentifizierungsvorgang abgebrochen.