Pemrograman Berbasis Web

Pertemuan 9

Nori Wilantika
Lya Hulliyyatus Suadaa
Yeni Rimawati

Politeknik Statistika STIS
Prodi DIV Komputasi Statistik

09

SERVER-SIDE SCRIPTING
<?PHP ?>

FORM HANDLING

POST

- Superglobals PHP $_POST digunakan untuk mengumpulkan data formulir dengan method=“post”.

```
<form action="welcome.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>
```

- Ketika pengguna mengisi formulir di atas dan mengklik tombol kirim, data formulir dikirim untuk diproses ke file PHP bernama "welcome.php". Data formulir dikirim dengan metode HTTP POST.

```
Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>
```

GET

- Superglobals PHP $_GET digunakan untuk mengumpulkan data formulir dengan method="get".

```
<html>
<body>
<a href="test_get.php?subject=PHP&web=W3schools.com">Test $GET</a>
</body>
</html>
```

- Berikut contoh kode pada test_get.php

```
<html>
<body>
<?php
echo "Study " . $_GET['subject'] . " at " . $_GET['web'];
?>
</body>
</html>
```

DATABASE CONNECTION

- With PHP, you can connect to and manipulate databases.
- MySQL is the most popular database system used with PHP.

- PHP 5 dan yang lebih baru dapat bekerja dengan database MySQL menggunakan:
  - Ekstensi MySQLi ("i" berarti improved)
  - PDO (PHP Data Object)
- Versi PHP sebelumnya menggunakan ekstensi MySQL. Namun, ekstensi ini tidak digunakan lagi sejak tahun 2012

- Baik MySQLi dan PDO memiliki kelebihan:
  - PDO akan bekerja pada 12 sistem database yang berbeda, sedangkan MySQLi hanya akan bekerja dengan database MySQL
  - Keduanya berorientasi objek, tetapi MySQLi juga menawarkan API prosedural.

CONNECTION

- Before we can interact with a DBMS we need to establish a connection to it
- A connection is established by creating an instance of the PDO class
- The constructor for the PDO class accepts arguments that specify **DSN (data source name), username, password, additional options**

```
$pdo = new PDO(dsn,username,password,options);
```

```
$conn = new PDO("mysql:host=$servername;dbname=myDB", $username, $password);
```

- PDO memerlukan database yang valid untuk disambungkan (di contoh: myDB). Jika tidak ada database yang ditentukan, exception akan dilempar.
- Koneksi ke database selain MySQL dilakukan dengan cara mengganti **prefix DSN (pgsql: , sqlsrv:)**
- The connection remains active for the lifetime of that PDO object. Koneksi akan ditutup secara otomatis saat skrip berakhir. Untuk menutup koneksi lebih awal, assigning NULL to the variable storing the PDO object destroys it

```
$pdo = NULL
```

```
$conn = null;
```

OPEN A CONNECTION

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";

try {
  $conn = new PDO("mysql:host=$servername;dbname=myDB", $username, $password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  echo "Connected successfully";
} catch(PDOException $e) {
  echo "Connection failed: " . $e->getMessage();
}
?>
```

https://www.php.net/manual/en/pdo.setattribute.php

QUERY

- The query() method of PDO objects can be used to execute an SQL query

```
$result = $pdo->query(statement)
$result = $pdo->query("SELECT␣*␣FROM␣meetings")
```

- query() returns the result set (if any) of the SQL query as a PDO Statement object

- The exec() method of PDO objects executes an SQL statement, returning the number of rows affected by the statement

```
$rowNum = $pdo->exec(statement)
$rowNum = $pdo->exec("DELETE␣*␣FROM␣meetings")
```

PROCESSING RESULT

- result set disimpan di sebuah PDOStatement object
- fetch() method digunakan untuk mengambil satu baris dari result. Hasilnya disimpan ke dalam sebuah array. We can use a while-loop together with the fetch() method to iterate over all rows in a result set.

```
while ($row = $result->fetch()) {
  echo "Slot:␣␣",$row["slot"], "<br>\n";
  echo "Name:␣␣",$row["name"], "<br>\n";
  echo "Email:␣",$row["email"],"<br><br>\n";
}
```

- fetchAll() method digunakan untuk mengambil seluruh baris dari result. Hasilnya disimpan ke dalam sebuah multidimensional array, then iterate over the array as often as you want.
- fetch: https://www.php.net/manual/en/pdostatement.fetch.php and fetchAll: https://www.php.net/manual/en/pdostatement.fetchall.php)
- By default, PDO returns each row as an array indexed by the column name and 0-indexed column position in the row, bisa diatur menggunakan setFetchMode().

INSERT

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $sql = "INSERT INTO MyGuests (firstname, lastname, email)
  VALUES ('John', 'Doe', 'john@example.com')";
  // use exec() because no results are returned
  $conn->exec($sql);
  echo "New record created successfully";
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

UPDATE & DELETE

```php
//Create new connection
$conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
// set the PDO error mode to exception
$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

//sql to select table
$result = $conn->query("UPDATE MyGuests SET lastname='Doe' WHERE id=2");

// echo a message to say the UPDATE succeeded
echo $result->rowCount() . " records UPDATED successfully";


// sql to delete a record
$sql = "DELETE FROM MyGuests WHERE id=3";

// use exec() because no results are returned
$conn->exec($sql);
echo "<br/><br/>Record deleted successfully";
```

FORM VALIDATION & SECURITY

FORM DATA

- Memeriksa kesesuaian isi field dengan format tertentu: PHP RegExp

```
$name = test_input($_POST["name"]);
if (!preg_match("/^[a-zA-Z-' ]*$/",$name)) {
  $nameErr = "Only letters and white space allowed";
}
```

```
$email = test_input($_POST["email"]);
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
  $emailErr = "Invalid email format";
}
```

- Memeriksa required field
  - `isset()` function checks whether a variable is set, which means that it has to be **declared AND is not NULL**
  - `empty()` function checks whether a variable is empty (0; 0.0; "0“; "“; NULL; FALSE; array( ))

TRANSACTION

- There are often situations where a single 'unit of work' requires a sequence of database operations; e.g., bookings, transfers

- To execute a sequence of SQL statements whose changes are
- only committed at the end once all have been successful or
- rolled back otherwise,

- PDO provides the methods
  - **beginTransaction()**
  - **commit()**
  - **rollBack()**

TRANSACTION

beginTransaction()

- By default, PDO runs in ”auto-commit” mode; successfully executed SQL statements cannot be 'undone’.
- Begintransaction() urns off auto-commit mode; changes to the database are not committed until commit() is called
- returns TRUE on success or FALSE on failure, throws an exception if another transaction is already active

commit()

- changes to the database are made permanent;
- returns TRUE on success or FALSE on failure, throws an exception if no transaction is active

rollBack()

- discard changes to the database; auto-commit mode is restored
- returns TRUE on success or FALSE on failure, throws an exception if no transaction is active

```php
try {
    //insert multiple rows
    // begin the transaction
    $conn->beginTransaction();
    // our SQL statements
    $conn->exec("INSERT INTO MyGuests (firstname, lastname, email)
    VALUES ('John', 'Doe', 'john@example.com')");
    $conn->exec("INSERT INTO MyGuests (firstname, lastname, email)
    VALUES ('Mary', 'Moe', 'mary@example.com')");
    $conn->exec("INSERT INTO MyGuests (firstname, lastname, email)
    VALUES ('Julie', 'Dooley', 'julie@example.com')");

    $last_id = $conn->lastInsertId();
    // commit the transaction
    $conn->commit();
    // use exec() because no results are returned

    //get last ID of increment
    echo "<br/>New records created successfully. Last inserted ID is: " . $last_id;
    }

catch(PDOException $e)
    {
    $conn->rollback();
    echo "Error: " . $e->getMessage();
    }
```

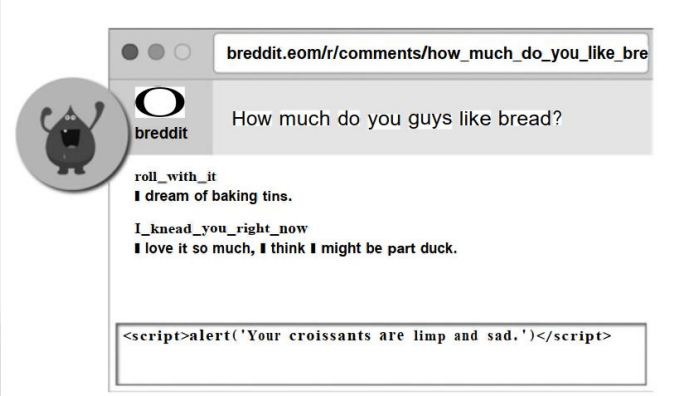
AVOID EXPLOITS: XSS

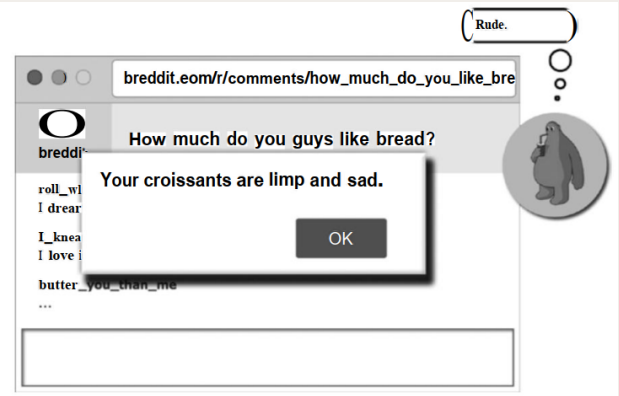
- client-side script injection: cross-site scripting (XSS). Contoh script yang diinject:

```
<script>location.href('http://www.hacked.com')</script>
```

```
<div style=position:absolute;top:0;bottom:0;left:0;right:0;bac
kground-color:black;font-size:100px;color:red;text-align:cente
r;>HAHAHAH ANDA TELAH DI HACK!!!!</div>
```

- XSS attack functions by taking advantage of the fact that web applications execute scripts on users' browsers. Ilustrasi:





AVOID EXPLOITS: XSS

- **`htmlspecialchars() function`**: mengubah karakter khusus menjadi entitas HTML.
- **`trim() function:`** menghapus karakter yang tidak perlu (spasi ekstra, tab, baris baru) dari data input pengguna
- **`stripslashes() function`**: menghapus backslashes (\) dari data input pengguna

```php
function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
```

```
<script>alert('hacked')</script>
```

```
&lt;script&gt;alert('hacked')&lt;/script&gt
```

AVOID EXPLOITS: SQL INJECTION

- SQL injection is the placement of malicious code in SQL statements, via web page input. the user gives you an SQL statement that you will unknowingly run on your database.
- SQL injection attacks, target websites that use an underlying SQL database and construct data queries to the database in an insecure fashion.

User Name:

105 OR 1=1

Password:

User Name:

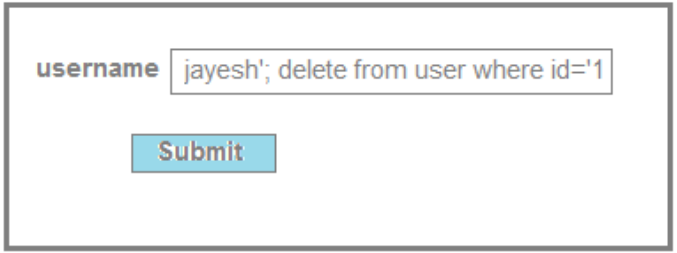
" or ""="

Password:

" or ""="

```
SELECT * FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""
```

```
SELECT UserId, Name, Password FROM Users WHERE UserId = 105 or 1=1;
```



PREPARED STATEMENTS

- To avoid SQL Injection, use Parameterized Statements: construct SQL strings using bind parameters.

- Using a prepared statement requires three steps
  1. Prepare: An SQL statement template is created and sent to the database. Certain values are left unspecified, called parameters (:name, where name is a PHP identifier, labeled "?").

```
$tpl1 = "select slot from meetings where
         name=:name and email=:email";
$tpl2 = "select slot from meetings where name=?";
```

  2. Binding the values to parameters: bindvalue() , bindparam()
  3. Executing the prepared statements, as many times as it wants with different values

- Binding of parameters to arguments will automatically prevent SQL injection.

```php
// prepare and bind
$stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email) VALUES (?, ?, ?)");
$stmt->bind_param("sss", $firstname, $lastname, $email);

// set parameters and execute
$firstname = "John";
$lastname = "Doe";
$email = "john@example.com";
$stmt->execute();

$firstname = "Mary";
$lastname = "Moe";
$email = "mary@example.com";
$stmt->execute();

$firstname = "Julie";
$lastname = "Dooley";
$email = "julie@example.com";
$stmt->execute();
```

AVOID EXPLOITS: FILE UPLOAD ATTACKS

Attackers can easily abuse file upload functions by injecting malicious code into an uploaded file. Web servers typically treat uploaded files like large blobs of binary data, so it’s pretty easy for an attacker to upload a malicious payload without the web server detecting it.

Mitigations:

- In the server side code, ensure uploaded files cannot be executed. consider adding some file-type checking.
- rename files and the extensions
- separating uploaded files into a particular directory or partition (so they aren’t intermingled with code)

AVOID EXPLOITS: BRUTE-FORCE ATTACKS

- Brute-Force Attacks: use scripts to try thousands of commonly used passwords (rainbow tables) against a login page.

123456	123456789	12345
Password	letmein	iloveyou
12345678	1234567	qwerty

Mitigations:

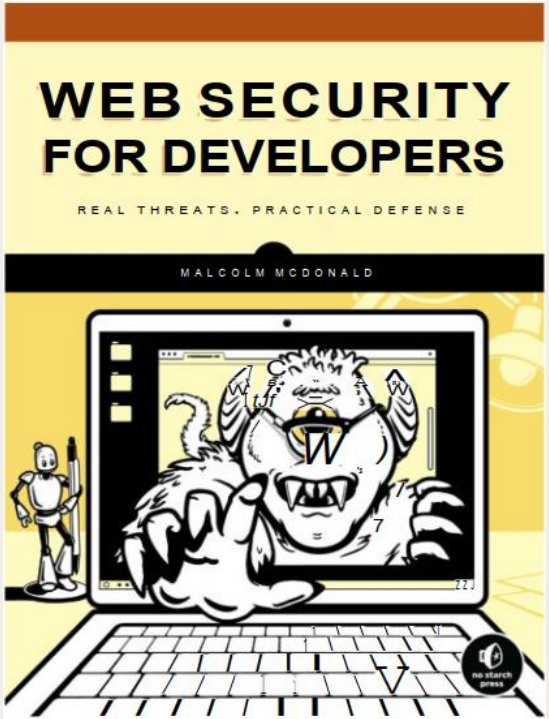
- Integrate with Single Sign-On (with social media or gmail account)
- Secure The Authentication System: Validating Email Addresses, Securing Password Resets, Requiring Complex Passwords
- Securely Storing Passwords: Do not simply store the password as is
  - Hashing Passwords: `hash(), md5()`
  - Salting Hashes: `md5(&password."X123!@#")`
- Requiring Multifactor Authentication (MFA)

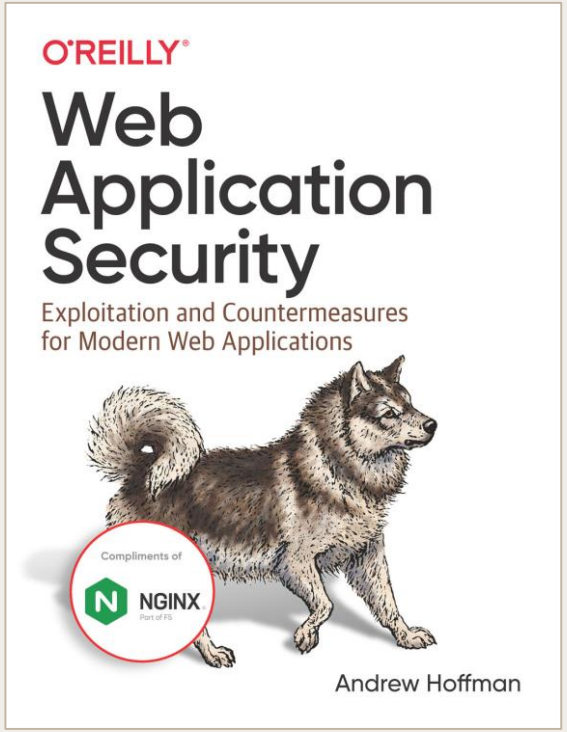
REFERENSI

Materi disadur dari:

1. COMP519 Web Programming (2020-21). Diakses pada Januari 2021, dari https://cgi.csc.liv.ac.uk/~ullrich/COMP519/
2. W3Schools Online Web Tutorials. Diakses pada Januari 2021, dari https://www.w3schools.com/

REFERENSI


WEB SECURITY
FOR DEVELOPERS
REAL THREATS. PRACTICAL DEFENSE
MALCOLM MCDONALD
no starch press


O'REILLY
Web Application Security
Exploitation and Countermeasures for Modern Web Applications
Compliments of
NGINX
Andrew Hoffman

Thanks!

Do you have any questions?

wilantika@stis.ac.id
lya@stis.ac.id
yeni.rima@stis.ac.id