Technische Universität Braunschweig

IAS | INSTITUTE FOR APPLICATION SECURITY

Exercise Sheet 6

Introduction to IT-Security

Submit your solutions via Gitlab
Deadline: *Wednesday, January 22nd, 09:00 a.m. CET 2026*

Basic Web Security

1. (4 points) Which of the following URLs satisfies the *same-origin policy* for `http://www.alice.org`? Explain your results.
   (a) (1 point) `http://www.alice.org:80/rotflcopter`
   (b) (1 point) `https://www.alice.org`
   (c) (1 point) `http://foo:lala@www.alice.org/swag`
   (d) (1 point) `http://yolo.alice.org`

2. (2 points) What is *cross-site request forgery*? Explain the goal of this attack type and how attacks are conducted. Which conditions need to be statisfied at the victim's side?

3. (2 points) Bob receives suspicious mails from Alice. When analyzing the mails for attacks he sees the following URLs. Which attack types can you identify?
   (a) (0.5 points) `http://tinyurl.com/cx9cecl`
   (b) (0.5 points) `http://foo.com/?q=%3CScrIpt%3Eal%65%52t%28%29%3C%2F%73c%52ipT%3E`
   (c) (0.5 points) `http://foo.com/?q=%27%20o%52%201%3D1%3B%20d%52%4Fp%20%74%62%6c`
   (d) (0.5 points) `http://foo.com/?q=%2e%2e%%35c%2e%2e%%35c%6c%6f%67`

SQL Injections (SQLi)

4. For this exercise we have provided a vulnerable server at `https://web.exercise.itsec.ias.tu-bs.de/`.

   (a) (2 points) A web server running on `https://web.exercise.itsec.ias.tu-bs.de/sqli/bachelor.php` uses the following PHP code as a login function. Specify login parameters that exploit the existing SQL injection vulnerability to login as `admin` and submit them in `src/sqli-bachelor.yml`

```php
<?php
$db = new SQLite3("mysqlitedb.db");

if ($_SERVER["REQUEST_METHOD"] === "POST") {
    $username = $_POST["username"] ?? "";
    $password = $_POST["password"] ?? "";

    $query = "SELECT * FROM users WHERE username = '$username' AND password = '$password'";

    $error = null;
    $result = @$db->query($query);
    if ($result === false) {
      $error = $db->lastErrorMsg();
    }
  }
?>

<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Bachelor SQLi</title>
    </head>
    <body>
        <div class="login-box">
            <h1>ITSEC SQLI Bachelor</h1>

    <?php if (isset($query)) {
     if ($result != false) {
       $user = $result->fetchArray(SQLITE3_ASSOC);

       if ($user) {
             if ($username === "student") {
               echo "<h2>Welcome, " . htmlspecialchars($user["username"]) . "!</h2>";
               echo "<p>You're logged in as a regular user. Try login as an admin.</p>";
              } else {
               if ($username !== "admin") {
                     echo "<h2>Injection into username field not possible.</h2>";
                    } else {
                     echo "<h2>Success! You logged in as admin!</h2>";
                    }
               }
            } else {
             echo "<h2>Login failed!</h2>";
             echo "<p>Invalid username or password.</p>";
            }
      }
    } else {
     echo "<p>Can you login as admin?</p>";
    } ?>

            <form method="POST">
                <label for="username">Username:</label>
                <input type="text" id="username" name="username" required>

                <label for="password">Password:</label>
                <input type="text" id="password" name="password" required>

                <button type="submit">Login</button>
            </form>
        </div>
    </body>
</html>
```

(b) (4 points) **Master:** web server running on `https://web.exercise.itsec.ias.tu-bs.de/sqli/master.php` uses the following PHP code as a product search function. Specify a search parameter that exploits the existing SQL injection vulnerability to find the hidden flag and submit the parameter in `src/sqli-master.yml`. Can you leak a secret that is not in the queried table?

```php
<?php
$db = new SQLite3("mysqlitedb.db");

if (isset($_GET["search"])) {
    $search = $_GET["search"];
    $query = "SELECT * FROM `products` WHERE `name` LIKE '%$search%'";
} else {
    $query = "SELECT * FROM `products` ORDER BY `name` ASC LIMIT 10";
}

$error = null;
$result = @$db->query($query);
if ($result === false) {
    $error = $db->lastErrorMsg();
}
?>
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Master SQLi</title>
    </head>
    <body>
        <div class="container">
            <h1>Products Search powered by SQLite</h1>

            <form method="GET">
                <input type="text" name="search" placeholder="Search products..."
                    value="<?= htmlspecialchars($_GET["search"] ?? "") ?>" />
                <input type="submit" value="Search" />
            </form>

            <table>
                <thead>
                    <tr>
                        <th>ID</th>
                        <th>Name</th>
                        <th>Price</th>
                    </tr>
                </thead>
                <tbody>
                    <?php if ($result): ?>
                        <?php while ($row = $result->fetchArray(SQLITE3_ASSOC)): ?>
                            <tr>
                                <td><?= htmlspecialchars($row["id"]) ?></td>
                                <td><?= htmlspecialchars($row["name"]) ?></td>
                                <td>$<?= htmlspecialchars($row["price"]) ?></td>
                            </tr>
                        <?php endwhile; ?>
                    <?php endif; ?>
                </tbody>
            </table>
        </div>
    </body>
</html>
```

Cross Site Scripting (XSS)

5. (2 points) Describe the differences between *reflected* and *stored* cross-site scripting attacks.

6. (10 points) Bob is running a web service at `https://web.exercise.itsec.ias.tu-bs.de/xss/`. The web service makes use of the GET parameters `var0`, ..., `var4` in the URL.

   Develop reflected cross-site scripting attacks for each parameter. Note that Bob uses different

sanitization techniques for each parameter. Show your success by opening a `alert()`. Provide your attacks as *URLs*. Attack URLs should be stored in single files `src/xss-var0.url` contains for example `https://web.exercise.itsec.ias.tu-bs.de/xss/?var0=[YOUR_PAYLOAD]`, ..., `src/xss-var4.url`. Each file hence contains only one line.

Hint:

1. Some web browsers block cross-site scripting attacks in URLs. Disable the protection for your particular browser if needed.
2. The injection of a simple script with a call of the `alert` method is sufficient to demonstrate the success of the attack.