

BEZPIECZEŃSTWO APLIKACJI PHP + MySQL

Zapytania przygotowane, SQL Injection oraz XSS na przykładzie formularza klienta

Spis treści

1. Struktura bazy danych
2. Kod pliku wiosna.php (formularz i zapis — wersja bezpieczna)
3. Kod pliku pokaz.php (wyświetlanie danych — wersja bezpieczna)
4. Wyjaśnienie: czym jest SQL Injection i jak chroni przed nim przygotowane zapytanie
5. Wyjaśnienie: jak działa przygotowanie zapytania krok po kroku
6. Wyjaśnienie: czym jest wstrzyknięcie kodu JavaScript (XSS) i jak chroni przed nim htmlspecialchars()
7. Podsumowanie zabezpieczeń

1. Struktura bazy danych

Wszystkie przykłady w tym materiale korzystają z tabeli klient o następującej strukturze:

```
CREATE TABLE klient (  
    id_klient INT AUTO_INCREMENT PRIMARY KEY,  
    imie VARCHAR(50) NOT NULL,  
    nazwisko VARCHAR(50) NOT NULL,  
    plec VARCHAR(1) NOT NULL,  
    miasto VARCHAR(50) NOT NULL,  
    data_zapisu DATE NOT NULL,  
    ulubiona_liczba INT NOT NULL  
);
```

2. Kod pliku wiosna.php (wersja bezpieczna)

```
<?php  
$conn = mysqli_connect("localhost","root","","wiosna_4g");  
  
if (!$conn){  
    die("błąd połączenia: " . mysqli_connect_error());  
}  
  
if (isset ($_POST['dodaj'])){  
    $imie = $_POST['imie'];  
    $nazwisko = $_POST['nazwisko'];  
    $plec = $_POST['plec'];  
    $miasto = $_POST['miasto'];
```

```

$data_zapisu = $_POST['data_zapisu'];
$ulubiona_liczba = rand(1, 100);

$dzisiaj = date("Y-m-d");

if ($data_zapisu > $dzisiaj){
    echo "Data zapisu nie może być z przyszłości!";
}
else{
    $z1 = "INSERT INTO `klient` (`id_klient`, `imie`, `nazwisko`, `plec`, `miasto`,
`data_zapisu`, `ulubiona_liczba`)
        VALUES (NULL, ?, ?, ?, ?, ?, ?)";

    $st = mysqli_prepare($conn, $z1);
    mysqli_stmt_bind_param($st, "sssssi", $imie, $nazwisko, $plec, $miasto,
$data_zapisu, $ulubiona_liczba);
    $rez = mysqli_stmt_execute($st);

    if ($rez){
        echo "dodałeś rekord: " . htmlspecialchars($_POST['nazwisko']) . "
(wylosowana liczba: " . $ulubiona_liczba . ")";
    }
    else{
        echo "nie udało się wstawić danych.";
    }

    mysqli_stmt_close($st);
}
}
?>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Formularz</title>
</head>
<body>

<form method="post">
    <label> Imię: </label>
    <input type="text" name="imie" required> <br>
    <label> Nazwisko: </label>
    <input type="text" name="nazwisko" required> <br>
    <label> Płeć: </label>
    <input type="radio" name="plec" value="k" required> Kobieta<br>
    <input type="radio" name="plec" value="m" > Mężczyzna<br>
    <label> Miasto: </label>
    <input type="text" name="miasto" required> <br>
    <label> Data zapisu: </label>
    <input type="date" name="data_zapisu" max="<?php echo date("\Y-m-d\"); ?>" required>
<br>

    <button type="submit" name="dodaj">Dodaj</button>
    <button type="reset">Reset</button>
</form>

<hr>

```

```
<form method="get" action="pokaz.php">
<button type="submit">Pokaż dane</button>
</form>

</body>
</html>
```

3. Kod pliku pokaz.php (wersja bezpieczna)

```
<?php
$conn = mysqli_connect("localhost","root","","wiosna_4g");

if (isset($_POST['usun'])){
    $z0 = "TRUNCATE TABLE `klient`";
    mysqli_query($conn,$z0);
}

$z1 = "SELECT id_klient,`imie`,`nazwisko`,`plec`,`miasto`,`data_zapisu`,`ulubiona_liczba`
FROM `klient`";
$rez = mysqli_query($conn,$z1);

if (mysqli_num_rows($rez) > 0){

    echo "<table border='1'>";
    echo "<tr>
        <th> ID klienta </th>
        <th> Imię </th>
        <th> Nazwisko </th>
        <th> Płeć </th>
        <th> Miasto </th>
        <th> Data zapisu </th>
        <th> Ulubiona liczba </th>
    </tr>";

    while ($row = mysqli_fetch_assoc($rez)){
        echo "<tr>";
        echo "<td>".htmlspecialchars($row['id_klient'])."</td>";
        echo "<td>".htmlspecialchars($row['imie'])."</td>";
        echo "<td>".htmlspecialchars($row['nazwisko'])."</td>";
        echo "<td>".htmlspecialchars($row['plec'])."</td>";
        echo "<td>".htmlspecialchars($row['miasto'])."</td>";
        echo "<td>".htmlspecialchars($row['data_zapisu'])."</td>";
        echo "<td>".htmlspecialchars($row['ulubiona_liczba'])."</td>";
        echo "</tr>";
    }
    echo "</table><br>";

    $z2 = "SELECT SUM(ulubiona_liczba) AS suma, AVG(ulubiona_liczba) AS srednia FROM
`klient`";
    $rez2 = mysqli_query($conn,$z2);
    $staty = mysqli_fetch_assoc($rez2);

    echo "<p>Suma wartości kolumny ulubiona_liczba: " . htmlspecialchars($staty['suma'])
. "</p>";
    echo "<p>Średnia wartość kolumny ulubiona_liczba: " .
htmlspecialchars(round($staty['srednia'], 2)) . "</p>";
```

```

}
else{
    echo "Brak rekordów do wyświetlenia.";
}

echo '<a href="wiosna.php">Powrót do formularza</a>';
?>

<form method="post">
    <button type="submit" name="usun">Usuń zawartość tabeli</button>
</form>

```

4. Czym jest SQL Injection?

SQL Injection (wstrzyknięcie kodu SQL) to atak polegający na tym, że osoba wypełniająca formularz wpisuje w pole tekstowe nie zwykłe dane (imię, nazwisko), lecz fragment kodu SQL. Jeśli aplikacja skleci taki tekst bezpośrednio z treścią zapytania, baza danych wykona go jako część polecenia — a nie jako zwykłą wartość.

Przykład ataku na kod NIEZABEZPIECZONY

Gdyby zapytanie wyglądało tak (bez przygotowania):

```

$z1 ="INSERT INTO `klient` (`id_klient`, `imie`, `nazwisko`, `plec`, `miasto`,
`data_zapisu`, `ulubiona_liczba`)
VALUES (NULL, '$imie', '$nazwisko', '$plec', '$miasto', '$data_zapisu',
$ulubiona_liczba)";

```

a atakujący w polu „Nazwisko” wpisałby:

```

x', 'k', 'Kraków', '2026-01-01', 1); DROP TABLE klient; --

```

to ostateczne zapytanie SQL wysłane do bazy wyglądałoby tak, jakby zawierało DWA polecenia — pierwsze wstawiające śmieciowy rekord, drugie usuwające całą tabelę klient. Baza wykonałaby oba, ponieważ dla niej to po prostu tekst zapytania, którego nie odróżnia od danych.

Dlaczego przygotowane zapytanie to blokuje

W wersji bezpiecznej zapytanie zawiera znaki zapytania ? zamiast wklejonych zmiennych:

```

$z1 = "INSERT INTO `klient` (...) VALUES (NULL, ?, ?, ?, ?, ?, ?)";
$st = mysqli_prepare($conn, $z1);

```

Funkcja `mysqli_prepare()` wysyła tę strukturę do MySQL, zanim jakiegokolwiek dane od użytkownika w ogóle się pojawią. Serwer bazy danych zapamiętuje szablon zapytania i wie dokładnie, które miejsca są kolumnami, a które będą wartościami. Gdy później `mysqli_stmt_bind_param()` podepnie zmienne, MySQL potraktuje je WYŁĄCZNIE jako dane — nawet jeśli zawierają apostrofy, średniki czy słowa kluczowe SQL. Wpisanie `x'; DROP TABLE klient; --` w pole nazwisko spowoduje jedynie zapisanie tego dosłownego, dziwnego tekstu jako wartości kolumny nazwisko — baza nie wykona z niego żadnego dodatkowego polecenia.

5. Jak działa przygotowanie zapytania — krok po kroku

1. `mysqli_prepare($conn, $z1)` — wysyła do serwera MySQL treść zapytania z placeholderami `?` i otrzymuje w zamian uchwyt do przygotowanego zapytania (tzw. statement), zapisywany zwyczajowo do zmiennej `$st` (skrót od statement).
2. `mysqli_stmt_bind_param($st, "sssssi", ...)` — podpiną kolejne zmienne PHP pod kolejne znaki `?` w zapytaniu, jednocześnie informując MySQL o typie każdej wartości.
3. `mysqli_stmt_execute($st)` — dopiero teraz zapytanie (szablon + dane) jest faktycznie wykonywane na bazie danych.

Ciąg znaków "sssssi" określa typ każdego parametru, w kolejności występowania:

Pozycja ?	Litera typu	Zmienna	Znaczenie litery
1	s	<code>\$imie</code>	string (tekst)
2	s	<code>\$nazwisko</code>	string (tekst)
3	s	<code>\$plec</code>	string (tekst)
4	s	<code>\$miasto</code>	string (tekst)
5	s	<code>\$data_zapisu</code>	string (tekst — MySQL sam zamienia na typ DATE)
6	i	<code>\$ulubiona_liczba</code>	integer (liczba całkowita)

Dostępne litery typów w mysqli

s — string (tekst), i — integer (liczba całkowita), d — double (liczba zmiennoprzecinkowa), b — blob (dane binarne, np. plik).

6. Czym jest wstrzyknięcie kodu JavaScript (XSS)?

XSS (Cross-Site Scripting) to atak polegający na tym, że dane wpisane przez użytkownika w formularzu trafiają najpierw do bazy danych, a następnie są wyświetlane na stronie — bez żadnego oczyszczenia — jako część kodu HTML. Jeśli te dane zawierają fragment kodu JavaScript, przeglądarka odwiedzającego stronę wykona go tak, jakby był to normalny element strony.

Przykład ataku na kod NIEZABEZPIECZONY

Gdyby wyświetlanie w `pokaz.php` wyglądało tak (bez `htmlspecialchars`):

```
echo "<td>".$row['imie']."</td>";
```

a atakujący wcześniej wpisał w polu „Imię” w formularzu:

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

to taki tekst zostałby zapisany w bazie jako zwykła wartość kolumny imię, a następnie na `pokaz.php` przeglądarka wstawiłaby go dosłownie do struktury HTML strony. Ponieważ przeglądarka nie odróżnia „tekstu do wyświetlenia” od „kodu do wykonania”, potraktowałaby znacznik `<script>` jako prawdziwy, uruchamiany kod JavaScript — w tym przykładzie wywołujący okienko `alert()`, ale w praktyce mógłby to być kod kradnący dane sesji, przekierowujący na inną stronę lub podmieniający zawartość strony.

Dlaczego `htmlspecialchars()` to blokuje

W wersji bezpiecznej każda wartość pochodząca z bazy (lub z formularza) jest przepuszczana przez funkcję `htmlspecialchars()` przed wyświetleniem:

```
echo "<td>".htmlspecialchars($row['imie'])."</td>";
```

Funkcja ta zamienia znaki mające specjalne znaczenie w HTML na ich bezpieczne odpowiedniki tekstowe, m.in.:

Znak wejściowy	Co zwraca htmlspecialchars()	Dlaczego to ważne
<	<	przeglądarka nie rozpozna początku znacznika (np. <script>)
>	>	przeglądarka nie rozpozna końca znacznika
"	"	nie da się przerwać atrybutu HTML
'	'	nie da się przerwać atrybutu w apostrofach

Dzięki temu tekst <script>alert('zaatakowano')</script> zostanie wyświetlony na stronie jako zwykły, widoczny dla użytkownika napis — a nie jako wykonywalny kod. Przeglądarka „zobaczy” tylko tekst, a nie żywy znacznik <script>.

7. Podsumowanie zabezpieczeń

Zagrożenie	Zastosowane zabezpieczenie	Gdzie w kodzie
SQL Injection	mysqli_prepare() + mysqli_stmt_bind_param()	wiosna.php — zapytanie INSERT
XSS (wstrzyknięcie JS)	htmlspecialchars() na każdej wyświetlanej wartości	pokaz.php — tabela i statystyki; wiosna.php — komunikat po zapisie
Nielogiczna data (z przyszłości)	porównanie dat w PHP + atrybut max w polu date	wiosna.php
Wyświetlanie pustej tabeli	sprawdzenie mysqli_num_rows(\$rez) > 0	pokaz.php

Do zapamiętania

Zapytania SELECT i TRUNCATE w tym materiale nie przyjmują żadnych danych od użytkownika, dlatego nie wymagają mysqli_prepare() — ryzyko SQL Injection dotyczy wyłącznie miejsc, w których dane z formularza (\$_POST) trafiają do treści zapytania SQL. Analogicznie, ryzyko XSS dotyczy wyłącznie miejsc, w których dane pochodzące od użytkownika są wypisywane z powrotem na stronie HTML.