Laboratorium 10

Ataki na skrypty PHP cz. 2

W sprawozdaniu zamieść zrzuty ekranu z istotnych etapów oraz odpowiedzi na pytania.

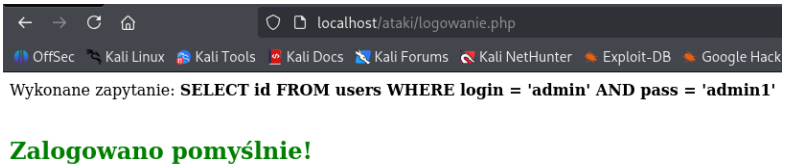
Taki symbol oznacza, że trzeba w sprawozdaniu dodać zrzut ekranu (najczęściej 1) z wyniku działania polecenie.

Taki symbol oznacza, że należy dodać opis (najczęściej 1 zdanie) z wyniku działania polecenia.

Niniejsze laboratorium stanowi kontynuację instrukcji z poprzednich zajęć, poświęconych zagadnieniom ataków na skrypty PHP.

I. Przygotowanie maszyny

1. Jeżeli tworzysz maszynę od początku, najpierw wykonaj poprzednią instrukcję do punktu 2.13.



II. SQL Injection – wyrażenie zawsze prawdziwe

Jak pamiętasz z ostatnich zajęć, gdy w polu hasła wpisałeś warunek logiczny: `cokolwiek' or 1 = '1`, nie musiałeś znać prawidłowego hasła, aby się zalogować.

To najpopularniejszy przykład ataku SQL Injection. Jego celem jest oszukanie bazy danych, aby "pomyślała", że podane hasło jest poprawne, nawet jeśli go nie znasz.

Standardowo zapytanie w kodzie PHP wygląda tak: `SELECT id FROM users WHERE login = '$log' AND pass = '$pas'`

Jeśli wpiszesz login `admin` i hasło `mojehaslo123`, baza wykona: `SELECT id FROM users WHERE login = 'admin' AND pass = 'mojehaslo123'`. Baza szuka wiersza, gdzie oba warunki są prawdziwe. Jeśli hasło jest złe, nic nie znajduje i nie loguje Cię.

Kiedy jednak w polu hasła wpiszesz frazę `cokolwiek' or 1 = '1`, "wstrzykujesz" dodatkową logikę do zapytania. Twoje wpisane hasło staje się częścią kodu SQL.

Zapytanie zmienia się w: `SELECT id FROM users WHERE login = 'admin' AND pass = 'cokolwiek' OR 1 = '1'`

Dlaczego to działa?

Baza danych analizuje warunki zgodnie z algebrą Boole'a. Zapytanie zostaje rozbite na dwie części oddzielone operatorem **OR** (LUB):

1. **Część A:** login = 'admin' AND pass = 'cokolwiek' (To jest **FAŁSZ**, bo hasło się nie zgadza).
2. **Część B:** 1 = '1' (To jest **PRAWDA**, bo jeden zawsze równa się jeden).

Ponieważ między nimi jest operator **OR**, całe wyrażenie staje się prawdziwe:

(FAŁSZ) LUB (PRAWDA) = PRAWDA

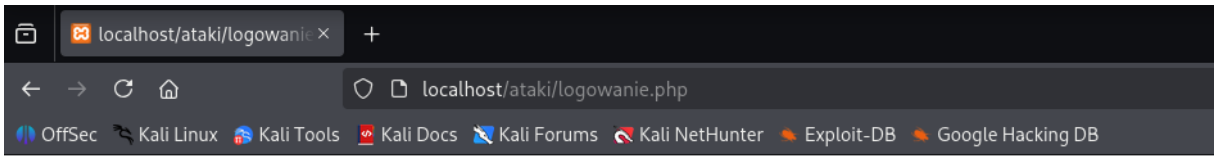
Dla bazy danych całe zapytanie jest teraz poprawne, więc zwraca rekord użytkownika admin, a skrypt PHP uznaje, że logowanie zakończyło się sukcesem.

Aby go zabezpieczyć powinno się przefiltrować dane wprowadzane do bazy. Do tego celu należy użyć funkcji **`mysqli_real_escape_string()`**, która dodaje znaki unikowe w łańcuchu znaków do użycia w instrukcji SQL.

1. W sekcji, gdzie odbierasz dane z formularza, musisz przepuścić zmienne przez tę funkcję. Podmień te dwie linie w swoim pliku:

```
// Odbieranie danych z formularza - Z ZASTOSOWANIEM ZABEZPIECZENIA
$log = mysqli_real_escape_string($link, $_POST['login']);
$pas = mysqli_real_escape_string($link, $_POST['pass']);
```

2. Spróbuj kolejnej próby ataku wspomnianą metodą. Powinno się nie udać.



Wykonane zapytanie: **SELECT id FROM users WHERE login = 'admin' AND pass = 'cokolwiek\' or 1 = \'1'**

Błąd logowania

Zauważ znak **`\`** przed apostrofem. Dla bazy danych oznacza to: "szukaj użytkownika, który ma w haśle fizyczny znak apostrofu". Ponieważ nikt nie ma takiego hasła, atak się nie powiedzie i zobaczysz napis **`Błąd logowania`**.

III. Przygotowanie maszyny SQL Injection – komentowanie niewygodnej części zapytania

Rozdział trzeci opisuje metodę SQL Injection polegającą na komentowaniu niewygodnej części zapytania. Dane przesyłane są metodą **`GET`** (widoczne w pasku adresu).

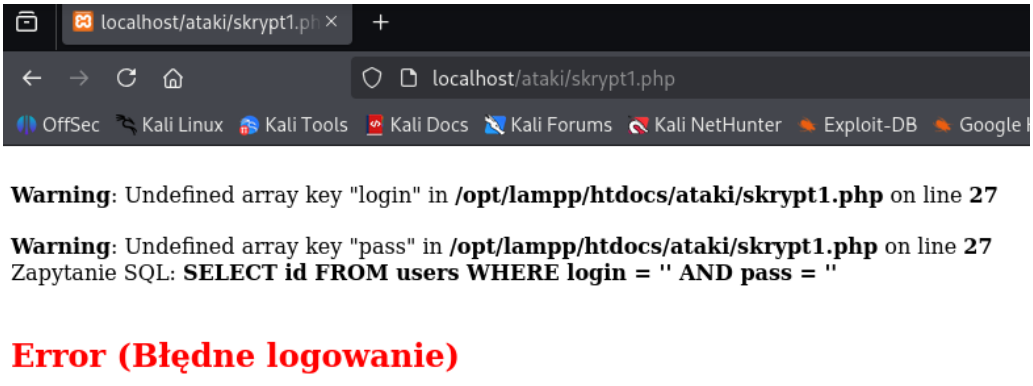
1. W katalogu **`/opt/lampp/htdocs/ataki`** utwórz plik **`skrypt1.php`**.

```
cd cd /opt/lampp/htdocs/ataki
sudo nano skrypt1.php
```

2. Wypełnij plik następującym kodem:

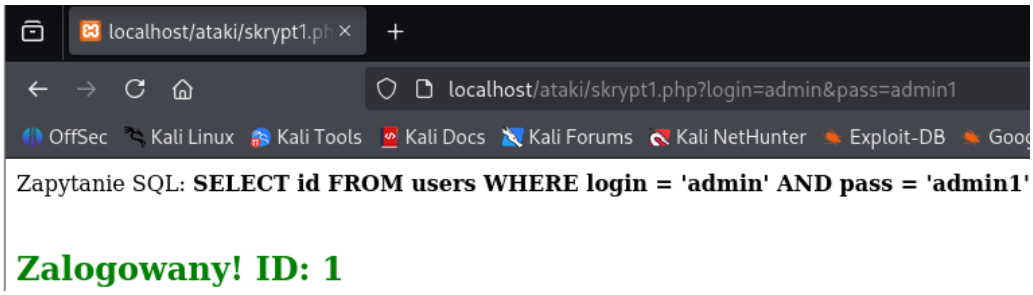
```php
<?php

// Włączanie raportowania błędów
ini_set('display_errors', 1);
error_reporting(E_ALL);

$hostname = 'localhost';
$user = 'root';
$password = ''; // Domyślnie puste w XAMPP
$database = 'testowa';

// Połączenie z bazą danych (MySQLi)
$link = mysqli_connect($hostname, $user, $password, $database);

if (!$link) {
    die("Błąd połączenia: " . mysqli_connect_error());
}

// Mechanizm automatycznego tworzenia zmiennych z parametrów GET
if (is_array($_GET)) {
    foreach ($_GET as $pos => $val) {
        // stripslashes usuwa ewentualne ukośniki dodane przez system
        $$pos = stripslashes(str_replace('\0', urldecode('%00'), $val));
    }
}

// Zapytanie SQL - UWAGA: Podatne, bo zmienne $_GET nie są filtrowane
$query = "SELECT id FROM users WHERE login = '".$_GET['login']."' AND pass =
'".$_GET['pass']."'";
// Wyświetlenie zapytania na stronie
echo "Zapytanie SQL: <b>" . $query . "</b><br><br>";
$result = mysqli_query($link, $query);
$txt = mysqli_fetch_array($result);
if (isset($txt['id'])) {
    echo "<h2 style='color:green'>Zalogowany! ID: " . $txt['id'] . "</h2>";
} else {
    echo "<h2 style='color:red'>Error (Błędne logowanie)</h2>";
}

mysqli_close($link);

?>
```

3. Uruchom skrypt w przeglądarce pod adresem: http://localhost/ataki/skrypt1.php.



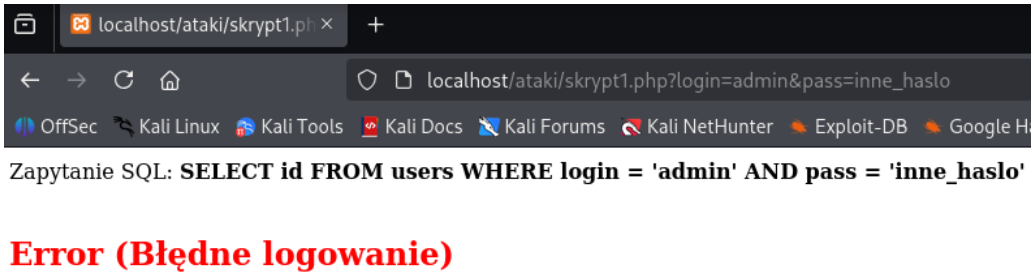
Standardowo zobaczysz błąd Error, ponieważ login i hasło są puste.

4. Z zapytania SQL bezpośrednio wynika, że strona pobiera login i hasło w celu zalogowania. Zaloguj się poprawnie: http://localhost/ataki/skrypt1.php?login=admin&pass=admin1.



Jak widać udało się zalogować. Co się stanie jednak jeżeli nie znasz hasła?

5. Wpisz: `http://localhost/ataki/skrypt1.php?login=admin&pass=inne_haslo`.

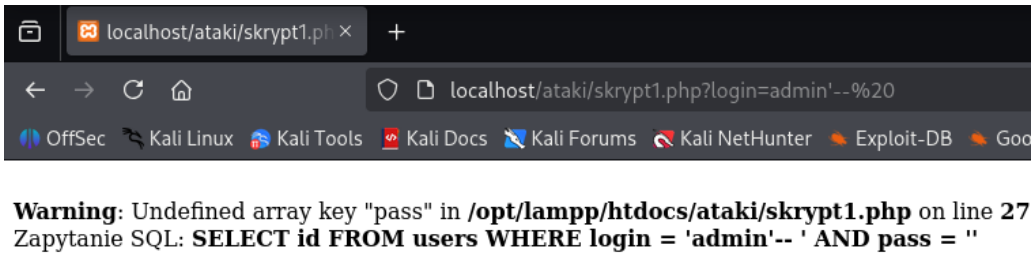


W konwencjonalny sposób zalogowanie się do systemu bez znajomości poprawnego hasła jest niemożliwe. Jednakże, dzięki podatności skryptu na ataki typu SQL Injection, możemy zmodyfikować strukturę zapytania SQL tak, aby pominąć proces weryfikacji hasła.

Naszym celem jest uzyskanie zapytania, które zweryfikuje jedynie login użytkownika: `SELECT id FROM users WHERE login = 'admin'`

Wykonaj atak (komentowanie hasła): Załóżmy, że znasz tylko login `admin`. Aby przejąć to konto, musisz skonstruować adres URL, który "odetnie" sprawdzanie hasła. W tym celu użyjemy apostrofu do zamknięcia frazy admin oraz sekwencji `--` ze spacją na końcu (zapis `%20` w adresie URL to nic innego jak techniczna reprezentacja spacji).

6. Wpisz w pasek adresu: http://localhost/ataki/skrypt1.php?login=admin'--%20



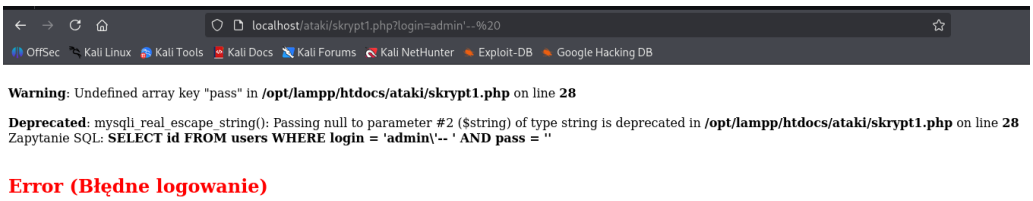
Zabezpieczenie przed atakiem wykorzystującym komentarze jest analogiczne do poprzedniego przykładu. Podstawową zasadą bezpieczeństwa jest nigdy nie ufać danym przesyłanym przez użytkownika. Wszystkie dane odbierane metodami GET lub POST muszą zostać przefiltrowane przed

umieszczeniem ich w zapytaniu SQL. W tym celu należy ponownie użyć funkcji zabezpieczającej **`mysqli_real_escape_string()`**.

7. Aby zabezpieczyć skrypt, zmodyfikuj linie odpowiedzialne za odbieranie danych z adresu URL (tablica $_GET):

```
// Połączenie z bazą musi być już nawiązane ($link)
$log = mysqli_real_escape_string($link, $_GET['login']);
$pas = mysqli_real_escape_string($link, $_GET['pass']);

// Teraz zapytanie jest bezpieczne
$query = "SELECT id FROM users WHERE login = '$log' AND pass = '$pas'";
```

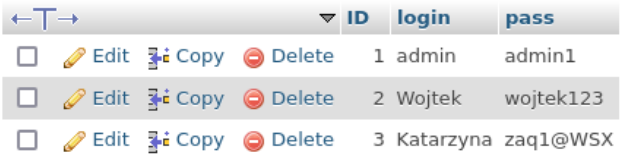
8. Spróbuj ponownie wywołać w przeglądarce adres:
`http://localhost/ataki/skrypt1.php?login=admin'--%20`



IV. SQL Injection – Union Select

Rozdział 4 instrukcji wprowadza Cię w znacznie groźniejszą odmianę ataku SQL Injection, która wykorzystuje operator **UNION**. W poprzednich krokach tylko omijałeś logowanie – teraz nauczysz się, jak zmusić bazę danych do "wyplucia" informacji, które normalnie są ukryte (np. wersja serwera, nazwy tabel, a nawet hasła innych użytkowników).

1. Na potrzeby kolejnego przykładu dodaj nowe rekordy do tabeli **`users`** w bazie danych:

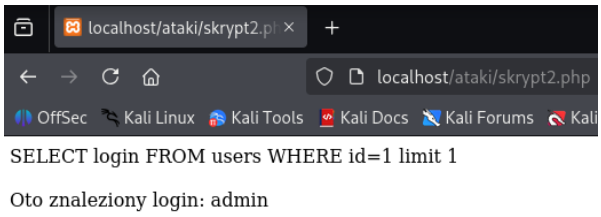


2. Utwórz plik **`skrypt2.php`** i wypełnij go kodem:

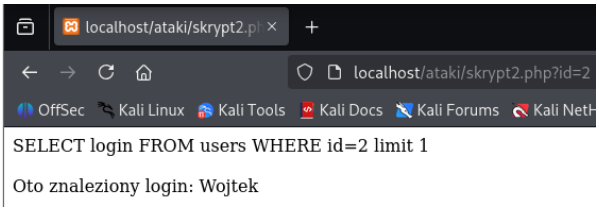
```
<?php

// Dane do połączenia z bazą danych MySQL
$hostname = 'localhost';
$user = 'root';
$password = '';
$database = 'testowa';

// Połączenie z bazą
$handle = mysqli_connect($hostname, $user, $password, $database) or die("Brak
połączenia z bazą danych");

// Mechanizm automatycznego tworzenia zmiennych z parametrów GET
if (is_array($_GET)) {
    foreach ($_GET as $pos => $val) {
        // Przetwarzanie znaków specjalnych
```

```
        $$pos = stripslashes(str_replace('\0', urldecode('100'), $val));
    }
}

// Domyślna wartość id, jeśli nie została podana
if (!isset($id)) {
    $id = 1;
}

// Zapytanie SQL limitujące liczbę rekordów do 1
$query = "SELECT login FROM users WHERE id=$id limit 1";

echo ($query . '<br>');

// Wykonanie zapytania
$result = mysqli_query($handle, $query);

// Pobranie i wyświetlenie wyniku
$txt = mysqli_fetch_array($result);
echo('<br>Oto znaleziony login: ' . $txt['login']);

mysqli_close($handle);

?>
```

3. Uruchom skrypt w przeglądarce.



4. Sprawdź, kto znajduje się pod **id=2**:



Załóżmy teraz, że chcemy uzyskać hasła użytkowników. Jedynym miejscem, w którym możemy wstrzyknąć własny kod, jest parametr **?id=**. W tym celu posłużymy się metodą **UNION SELECT**.

W MySQL instrukcja **UNION** łączy wyniki dwóch zapytań w jedną tabelę. Warunkiem jej użycia jest to, aby obie części zapytania (przed i po **UNION SELECT**) posiadały taką samą liczbę kolumn.

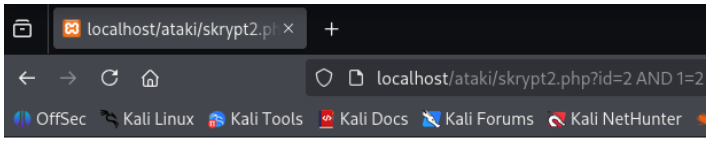
Na tym etapie, nawet jeśli dopiszemy **UNION SELECT** do zapytania w bazie danych w celu wyświetlenia hasła użytkownika, nadal zostanie wyświetlony wyłącznie login – wszystko przez zastosowanie klauzuli **LIMIT 1**.

Naszym celem jest więc pozbycie się pierwszej części zapytania. Można to osiągnąć poprzez celowe zafałszowanie warunku log

icznego, tak aby nigdy nie został spełniony.

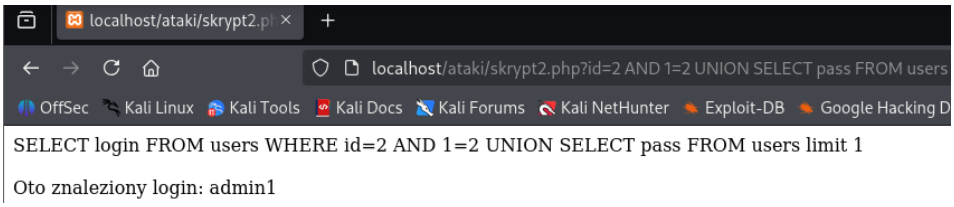
5. W tym celu dopisz: **http://localhost/ataki/skrypt2.php?id=2 AND 1=2**

Tak skonstruowane zapytanie nie zwraca żadnych wyników, ponieważ warunek logiczny jest zawsze fałszywy.



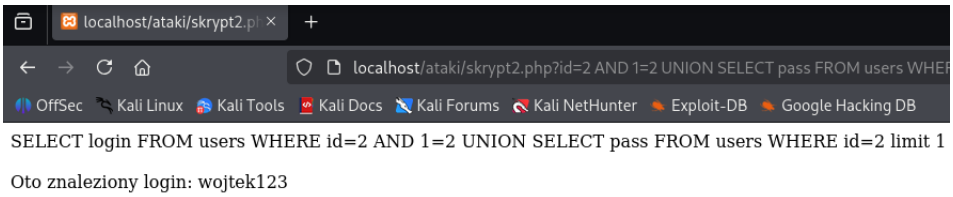
Jeżeli połączymy zapytanie, które nie zwraca żadnych wyników, z zapytaniem zwracającym dane, wówczas otrzymamy rezultat wyłącznie z drugiej części zapytania.

6. W tym celu dopisujemy: `?id=2 AND 1=2 UNION SELECT pass FROM users`



Po sprawdzeniu wyniku powinniśmy uzyskać hasło użytkownika – jednak nie użytkownika o `id=2`, lecz tego o `id=1`, czyli pierwszego rekordu w bazie danych.

7. Aby uzyskać hasło konkretnego użytkownika, wystarczy nieznacznie zmodyfikować zapytanie: `?id=2 AND 1=2 UNION SELECT pass FROM users WHERE id=2`



Jak widać, metoda ta pozwala na uzyskanie hasła dowolnego użytkownika znajdującego się w bazie danych. Technika `UNION SELECT` należy do jednych z najskuteczniejszych metod ataków typu SQL Injection, jednak również przed nią można się skutecznie zabezpieczyć.

W tym przypadku przekazywany parametr `id` powinien być liczbą, dlatego wystarczy odpowiednio go przefiltrować, na przykład za pomocą funkcji `intval()`.

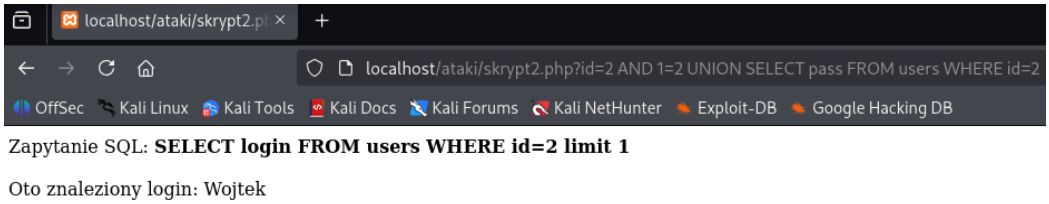
Funkcja `intval()` zamienia przekazany parametr na liczbę całkowitą. Jeżeli użytkownik spróbuje przekazać zapytanie SQL, funkcja zwróci wartość 0, co skutecznie uniemożliwi przeprowadzenie kolejnego ataku.

8. Zmodyfikuj kod w `skrypt2.php`:

```php
<?php

// Dane do połączenia z bazą danych
$hostname = 'localhost';
$user = 'root';
$password = '';
$database = 'testowa';
```

```php
// Połączenie z bazą danych
$handle = mysqli_connect($hostname, $user, $password, $database) or die("Brak
połączenia z bazą danych");

// Mechanizm automatycznego tworzenia zmiennych z parametrów GET
if (is_array($_GET)) {
    foreach ($_GET as $pos => $val) {
        $$pos = stripslashes(str_replace('\0', urldecode('%00'), $val));
    }
}

// Zastosowanie funkcji intval() przed wykonaniem zapytania
if (isset($id)) {
    $id = intval($id);
}

// Zapytanie SQL z użyciem przefiltrowanej zmiennej $id
$query = "SELECT login FROM users WHERE id=$id limit 1";

echo "Zapytanie SQL: <b>" . $query . "</b><br>";

// Wykonanie zapytania
$result = mysqli_query($handle, $query);
$txt = mysqli_fetch_array($result);

// Wyświetlenie wyniku
echo '<br>Oto znaleziony login: ' . $txt['login'];

mysqli_close($handle);

?>
```

9. Wykonaj ponownie ppkt. 7.



Zapytanie SQL: **SELECT login FROM users WHERE id=2 limit 1**

Oto znaleziony login: Wojtek

Dzięki tej modyfikacji, próba ataku **`UNION SELECT`** kończy się niepowodzeniem, ponieważ baza danych otrzyma zapytanie o treści **`WHERE id=2`**, ignorując całkowicie wstrzyknięte polecenia.