



Politechnika Bydgoska
im. Jana i Jędrzeja Śniadeckich
Wydział Telekomunikacji, Informatyki i Elektrotechniki
Zakład Techniki Cyfrowej



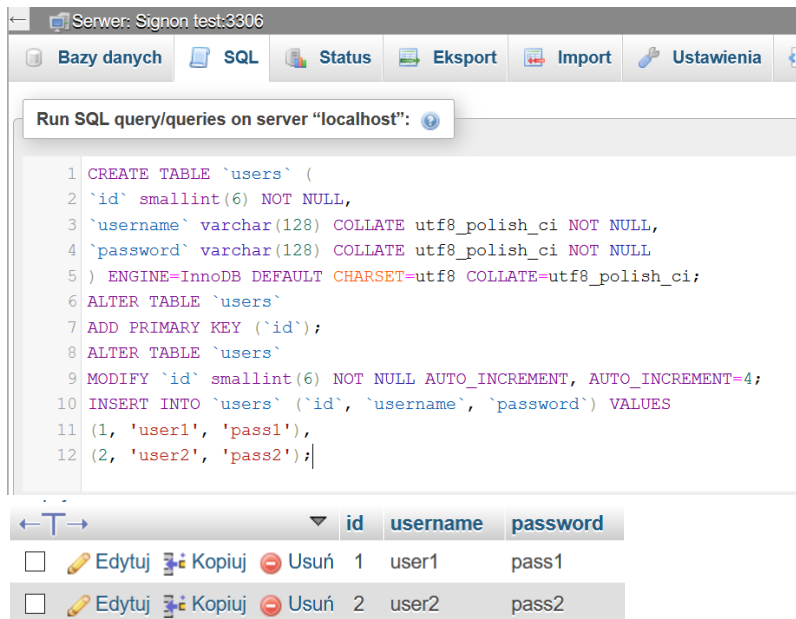
Przedmiot:	Programowanie aplikacji sieciowych		Teleinformatyka Studia stacjonarne Semestr 5, 2021/2022
Temat:	Audyt bezpieczeństwa aplikacji sieciowych (<i>SQL Injection, Brute Force</i>).		
Numer lab.:	5	Data wykonania:	2021.11.09
Prowadzący:	dr inż. Piotr Grad	Data oddania:	2021.11.10
Autor:	Anna Bagniewska	Indeks:	114881

1. Opis zadania

Zadanie polega na uświadomieniu odbiorcom zagrożeń wynikających ze słabo zabezpieczonych kont i witryn. Jednymi z zagrożeń są SQL Injections oraz Brute Force. W zadaniu należy utworzyć proste formularze logowania i sprawdzić ich podatność na ataki. Następnie należy usprawnić systemy zabezpieczeń i ponownie przetestować ochronę.

2. Opis programu

Utworzenie tabeli w bazie danych:



The screenshot shows a MySQL database interface with the following SQL query executed:

```
1 CREATE TABLE `users` (  
2   `id` smallint(6) NOT NULL,  
3   `username` varchar(128) COLLATE utf8_polish_ci NOT NULL,  
4   `password` varchar(128) COLLATE utf8_polish_ci NOT NULL  
5 ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_polish_ci;  
6 ALTER TABLE `users`  
7 ADD PRIMARY KEY (`id`);  
8 ALTER TABLE `users`  
9 MODIFY `id` smallint(6) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=4;  
10 INSERT INTO `users` (`id`, `username`, `password`) VALUES  
11 (1, 'user1', 'pass1'),  
12 (2, 'user2', 'pass2');
```

Below the query, a table view shows the data:

	id	username	password
<input type="checkbox"/>	1	user1	pass1
<input type="checkbox"/>	2	user2	pass2

Stworzenie prostego formularza logowania z wykorzystaniem metody POST:

```
Formularz logowania  
<form method="post" action="weryfikuj1.php">  
  Login:<input type="text" name="user" maxlength="20" size="20"><br>  
  Hasło:<input type="password" name="pass" maxlength="20" size="20"><br>  
  <input type="submit" value="Send"/>  
</form>
```

Oraz prostego, podatnego na SQL Injection skryptu sprawdzającego dane logowania:

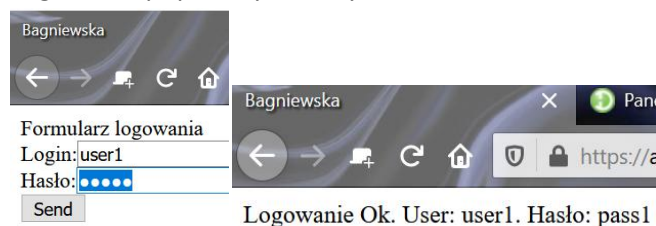
```
$user=$_POST['user'];// login z formularza  
$pass=$_POST['pass'];// hasło z formularza
```

Zapytanie podatne na atak:

```
$result = mysqli_query($link, "SELECT * FROM users WHERE (username='$user') and (password='$pass')");
```

Test działania skryptu:

- Logowanie poprawnymi danymi:



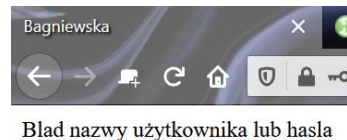
The screenshot shows a web browser window with the URL <https://a>. The page displays a login form with the following fields:

- Formularz logowania
- Login: user1
- Hasło: [password masked]
- Send button

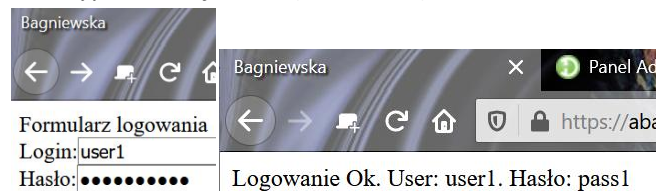
Below the form, the message "Logowanie Ok. User: user1. Hasło: pass1" is displayed.



- Logowanie częściowo poprawnymi danymi:
W każdym przypadku otrzymuje się poniższy rezultat:

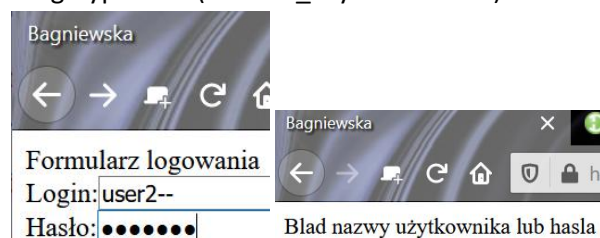


- Atak typu SQL Injection ('OR 1=1--'):



Potwierdza to brak zabezpieczeń przeciwko SQL Injection w skrypcie 1.

- Drugi typ ataku (<nazwa_użytkownika>--):



Skrypt na wybranym hostingu nie jest podatny na ten rodzaj ataku.

Skrypt 2:

Zawiera dodatkowe i uzupełnione polecenia:

```
$result = mysqli_query($link, "SELECT * FROM users WHERE username='$user'");  
if($rekord['password']==$pass)
```

Autoryzacja użytkowników działa poprawnie.

Test drugiego skryptu pod kątem SQL Injection:



Błąd w haśle !

Skrypt nie rozpoznał podanego hasła, bo nie ma go w bazie.

Weryfikacja podatności skryptu na atak wybrany z OWASP:

Podatne i przestarzałe komponenty są jedną ze ścieżek do atakowania skryptów. Należy sprawdzać na bieżąco wersje komponentów po stronie klienta i serwera, pozyskiwać komponenty programów z zaufanych źródeł oraz usuwać nieużywane zależności, niepotrzebne funkcje, komponenty, pliki i dokumentację. Są to tylko dwa przykłady z kilku, które podnoszą bezpieczeństwo aplikacji. Do przetestowania podatności od tej strony można wykorzystać np.: WASP Dependency Check lub retire.js.

Modyfikacja skryptu weryfikuj2.php z wykorzystaniem htmlentities():
Stwarza to problem, gdy w haśle będą występować inne znaki specjalne.

```
$user = htmlentities ($_POST['user'], ENT_QUOTES, "UTF-8");  
$pass = htmlentities ($_POST['pass'], ENT_QUOTES, "UTF-8");
```

Rozbudowanie pliku weryfikuj3.php o zmienne sesyjne:

```
<?php  
    session_start();  
?  
  
$_SESSION ['loggedin'] = true;
```

Fragment obecny we wszystkich skryptach:

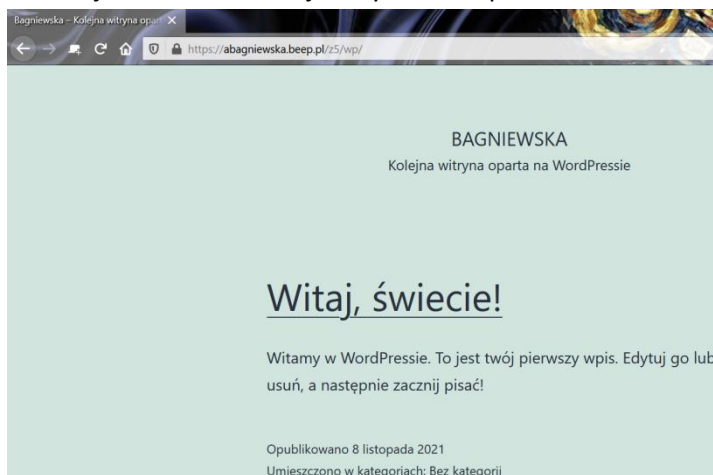
```
if (isset($_SESSION['loggedin'])){  
    header('Location: index4.php');  
}else{  
    header('Location: index3.php');  
}
```

Jeżeli w aktualnej sesji nie ma zmiennej sesyjnej loggedin == true, zostaje się przekierowanym do strony logowania. Ogranicza to możliwość wejścia na stronę bez logowania, mimo znajomości struktury strony.

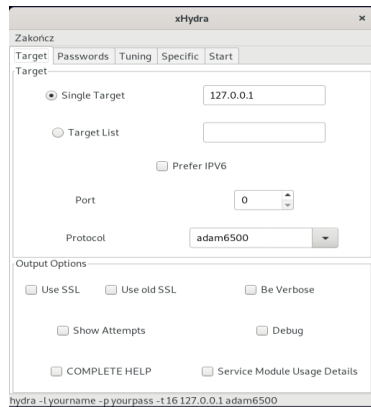
Utworzenie przycisku do wylogowania w index4.php i skrypt wylogowania:

```
<BODY>  
    <a href="logout.php">Wyloguj</a><br/>  
</body>  
  
<?php  
    session_start();  
    session_unset();  
    header('Location: index3.php');  
?>
```

Instalacja WordPress z najniższymi zabezpieczeniami:



Przykładowe narzędzie do przeprowadzania ataków/prób zabezpieczeń (xHydra):



Rezultat przy podanym loginie i stworzonym pliku z przykładowymi hasłami do sprawdzenia:

```
Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2021-11-08 15:55:13
[DATA] max 4 tasks per 1 server, overall 4 tasks, 4 login tries (l:1/p:4), ~1 try per task
[DATA] attacking http-get://193.239.44.226:80/https://abagniewska.beep.pl/z5/wp
[ATTEMPT] target 193.239.44.226 - login "abagniewska5wp" - pass "qwerty" - 1 of 4 [child 0] (0/0)
[ATTEMPT] target 193.239.44.226 - login "abagniewska5wp" - pass "1qaz@WSX" - 2 of 4 [child 1] (0/0)
[ATTEMPT] target 193.239.44.226 - login "abagniewska5wp" - pass "test" - 3 of 4 [child 2] (0/0)
[ATTEMPT] target 193.239.44.226 - login "abagniewska5wp" - pass "asdfghjkl" - 4 of 4 [child 3] (0/0)
[80][http-get] host: 193.239.44.226 login: abagniewska5wp password: 1qaz@WSX
<finished>
```

3. Wnioski:

Wprowadzając aplikację na rynek, należy pamiętać o usunięciu swoich komunikatów błędów, które w niewłaściwych rękach mogą zostać wykorzystane przeciwko nam.

Ponadto należy uważać na luki w zabezpieczeniach aplikacji. Z wykorzystaniem najprostszych metod można złamać zabezpieczenia nawet popularnych witryn.