Instrukcja 3

Ataki na skrypty PHP

Wstęp

Celem niniejszej instrukcji jest zapoznanie się z najczęstszymi podatnościami aplikacji webowych napisanym w języku PHP, takimi jak SQL Injection oraz Cross-Site Scripting (XSS). Poprzez praktyczne odwzorowanie błędów programistycznych w kontrolowanym środowisku, nauczysz się, w jaki sposób hakerzy mogą przejąć kontrolę nad bazą danych lub sesją użytkownika.

Atak SQL Injection polega na wstrzyknięciu złośliwego kodu do zapytania wysyłanego do bazy danych SQL. Dzięki wykorzystaniu znaków specjalnych możliwe jest modyfikowanie treści zapytania w taki sposób, aby ominąć mechanizmy zabezpieczeń aplikacji.

XSS to jeden z najpowszechniejszych błędów w aplikacjach webowych. Polega on na braku filtrowania znaczników HTML oraz skryptów JavaScript w danych przesyłanych przez użytkownika. Luka ta umożliwia atakującemu modyfikację źródła strony, co otwiera drogę do kradzieży ciasteczek (cookies), przekierowań na złośliwe witryny czy przejmowania sesji użytkowników.

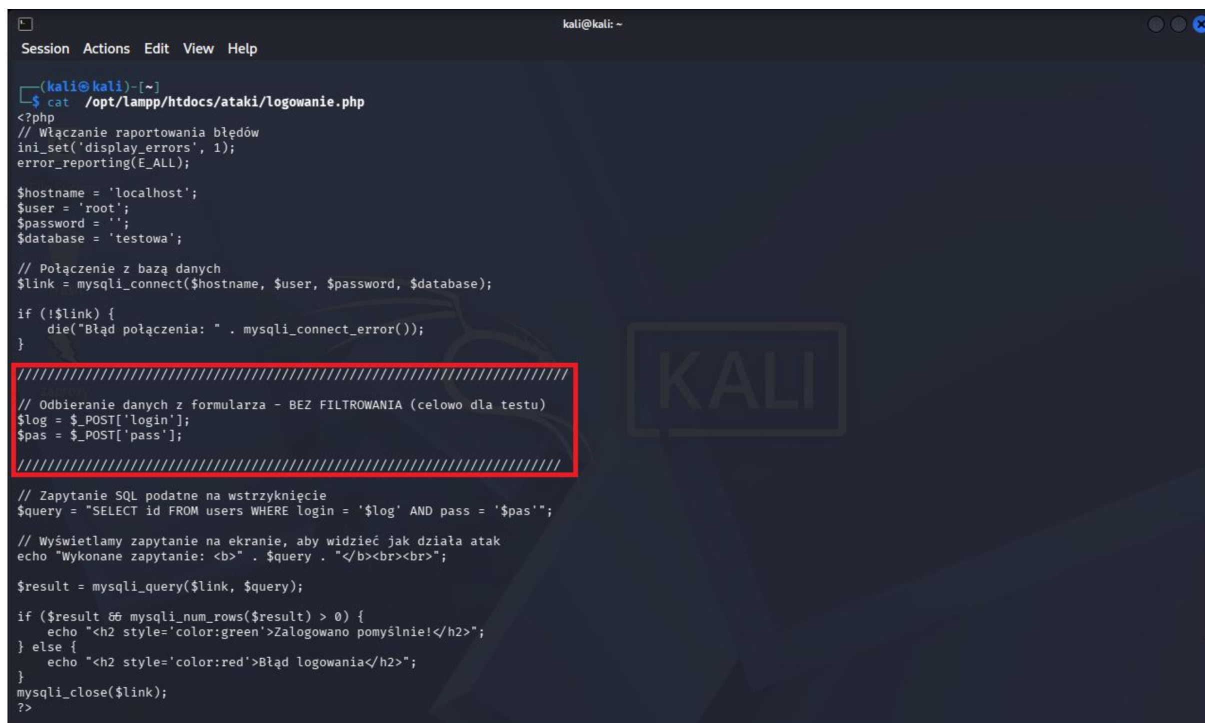
Ataki XSS najczęściej spotyka się w miejscach, gdzie użytkownik może zostawić trwały ślad tekstowy, takich jak księgi gości, systemy komentarzy czy fora.

I. SQL Injection

Na potrzeby zajęć przygotowano formularz logowania, który wywołuje skrypt obsługujący proces uwierzytelniania użytkownika.

1. Przejrzyj zawartość skryptu (zwróć szczególną uwagę na fragment odpowiedzialny za odbieranie danych logowania):

```
cat /opt/lampp/htdocs/ataki/logowanie.php
```

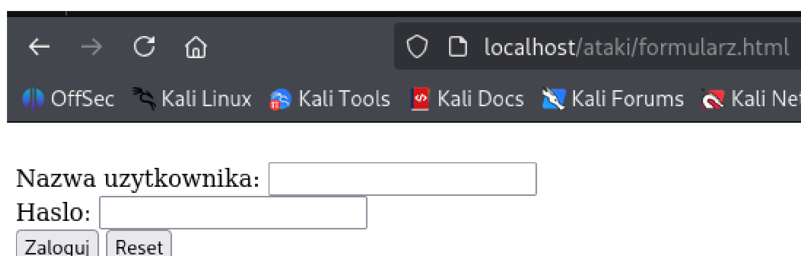


Po przesłaniu danych z pliku formularz.html, dane są przetwarzane przez skrypt logowania. Na początku skryptu definiowane są zmienne umożliwiające połączenie z przykładową bazą danych testowa, utworzoną na lokalnym serwerze przy użyciu programu XAMPP.

W dalszej części skrypt nawiązuje połączenie z bazą danych i wykonuje zapytanie SQL, którego celem jest pobranie ID użytkownika z tabeli `users`, którego pola `login` i `pass` odpowiadają wartościom przesłanym w formularzu.

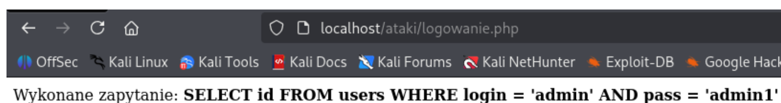
Na końcu sprawdzane jest, czy zapytanie zwróciło jakiekolwiek wyniki (czy liczba wierszy jest większa niż 0). Jeśli tak, użytkownik zostaje zalogowany. W przedstawionym przykładzie logowanie ogranicza się jedynie do wyświetlenia komunikatu tekstowego. W rzeczywistych zastosowaniach mogłoby to obejmować np. przyznanie odpowiednich uprawnień, ustawienie ciasteczek, przekierowanie na inną stronę lub konfigurację zmiennych sesji.

2. W przeglądarce wpisz adres: http://localhost/ataki/formularz.html



3. Spróbuj zalogować się, używając poprawnych danych:

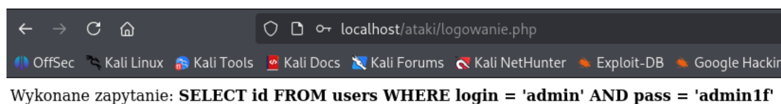
login: `admin`   hasło: `admin1`



Wykonane zapytanie: **SELECT id FROM users WHERE login = 'admin' AND pass = 'admin1'**

Zalogowano pomyślnie!

4. Spróbuj zalogować się ponownie, podając niepoprawne hasło.



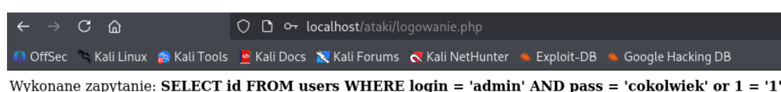
Wykonane zapytanie: **SELECT id FROM users WHERE login = 'admin' AND pass = 'admin1f'**

Błąd logowania

Skrypt działa poprawnie i zabezpiecza nas przed nieautoryzowanym dostępem, ale czy na pewno?

5. Sprawdź, co się stanie, gdy w polu hasła wpiszesz:

```
cokolwiek' or 1 = '1
```



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

Zalogowano pomyślnie!

W tym przypadku aplikacja zwróci komunikat świadczący o poprawnym zalogowaniu, mimo braku znajomości prawidłowego hasła! Jest to przykład skutecznie przeprowadzonego ataku SQL Injection.

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:

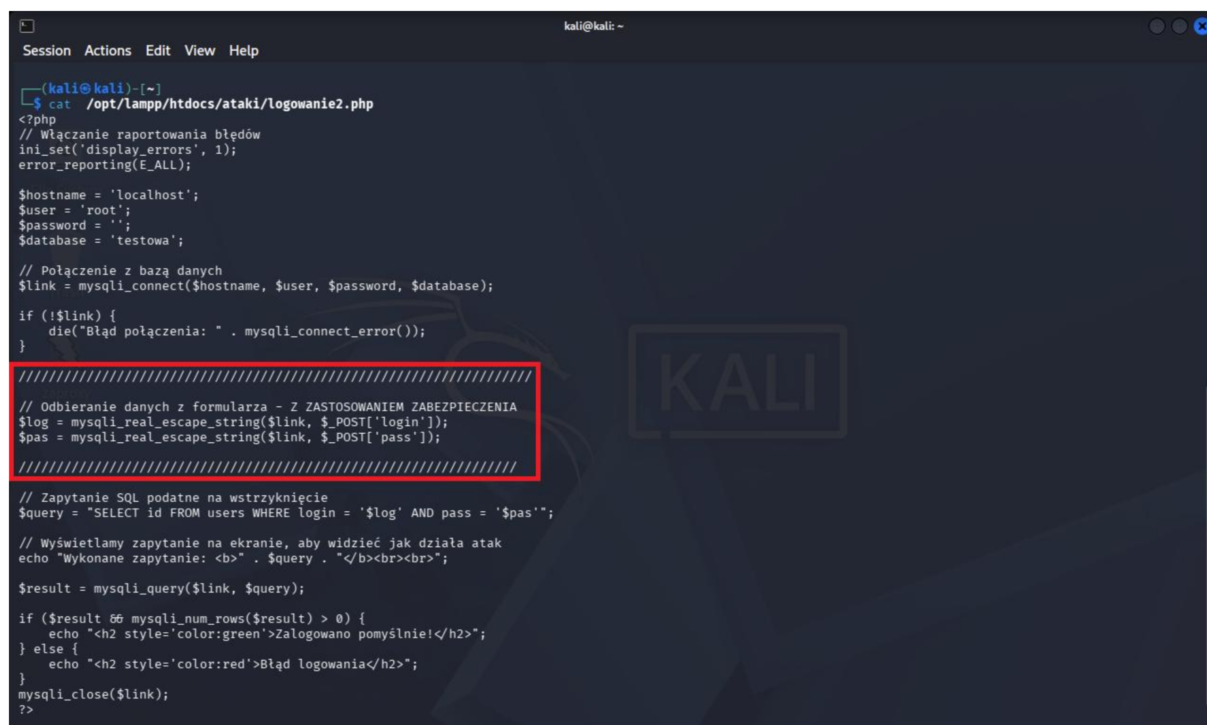
(FAŁSZ) LUB (PRAWDA) = PRAWDA

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

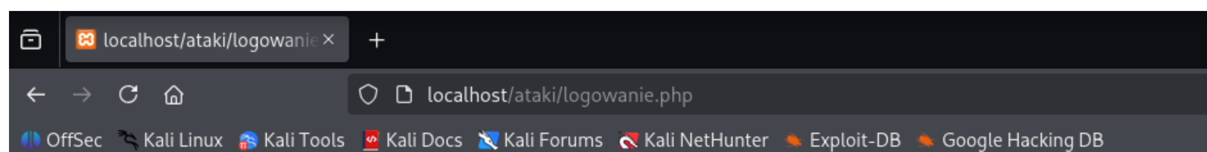
Aby go zabezpieczyć powinno się przefiltrować dane wprowadzane do bazy. Do tego celu należy użyć funkcji `mysqli_real_escape_string()`, która dodaje znaki unikowe w łańcuchu znaków do użycia w instrukcji SQL.

6. Przejrzyj zawartość zmodyfikowanego skryptu: (ponownie zwróć szczególną uwagę na fragment odpowiedzialny za przetwarzanie danych logowania):

```
cat /opt/lampp/htdocs/ataki/logowanie2.php
```



7. Spróbuj kolejnej próby ataku wspomnianą metodą. Tym razem atak nie powinien się powieść.



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`.

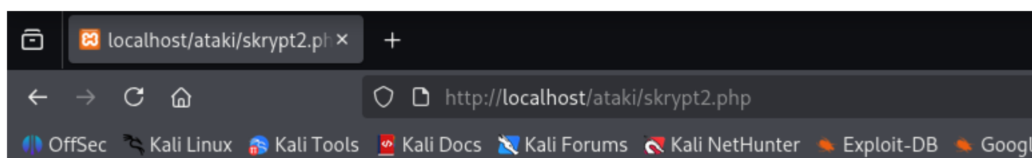
SQL Injection – komentowanie niewygodnej części zapytania

Kolejną metodą metodę SQL Injection polegaja na komentowaniu niewygodnej części zapytania. Dane przesyłane są metodą GET (widoczne w pasku adresu).

6. Przejrzyj zawartość skryptu z podatnością:

```
cat /opt/lampp/htdocs/ataki/skrypt2.php
```

7. Uruchom skrypt w przeglądarce pod adresem: http://localhost/ataki/skrypt2.php



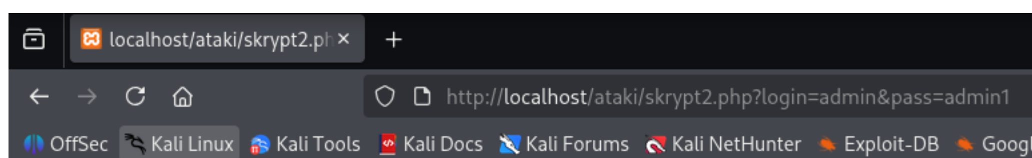
Warning: Undefined array key "login" in **/opt/lampp/htdocs/ataki/skrypt2.php** on line **27**

Warning: Undefined array key "pass" in **/opt/lampp/htdocs/ataki/skrypt2.php** on line **27**
Zapytanie SQL: **SELECT id FROM users WHERE login = '' AND pass = ''**

Error (Błędne logowanie)

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

8. Z zapytania SQL bezpośrednio wynika, że strona pobiera login i hasło w celu zalogowania. Zaloguj się poprawnie: http://localhost/ataki/skrypt2.php?login=admin&pass=admin1



Zapytanie SQL: **SELECT id FROM users WHERE login = 'admin' AND pass = 'admin1'**

Zalogowany! ID: 1

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

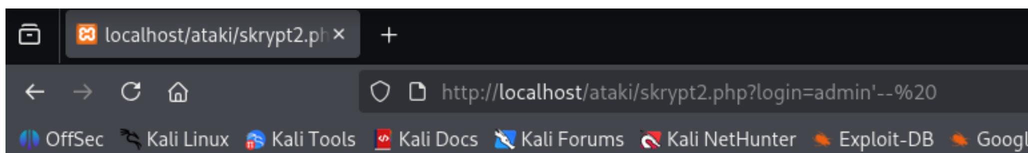
9. Wpisz: `http://localhost/ataki/skrypt2.php?login=admin&pass=inne_haslo`

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).

10. Wpisz w pasek adresu: http://localhost/ataki/skrypt2.php?login=admin'--%20



Warning: Undefined array key "pass" in **/opt/lampp/htdocs/ataki/skrypt2.php** on line **27**
Zapytanie SQL: **SELECT id FROM users WHERE login = 'admin'-- ' AND pass = ''**

Zalogowany! ID: 1

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()`.

11. Aby zabezpieczyć skrypt, zmodyfikuj linie odpowiedzialne za odbieranie danych z adresu URL (tablica $_GET). Wykorzystaj do tego celu edytor `nano`.

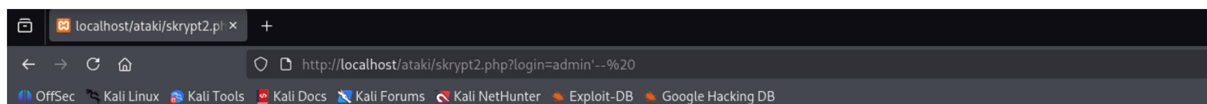
```
sudo nano /opt/lampp/htdocs/ataki/skrypt2.php
```

12. Zmodyfikuj skrypt.

```
// 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'";
```

13. Spróbuj ponownie wywołać w przeglądarce adres: `http://localhost/ataki/skrypt2.php?login=admin'--%20`



Warning: Undefined array key "pass" in **/opt/lampp/htdocs/ataki/skrypt2.php** on line **28**

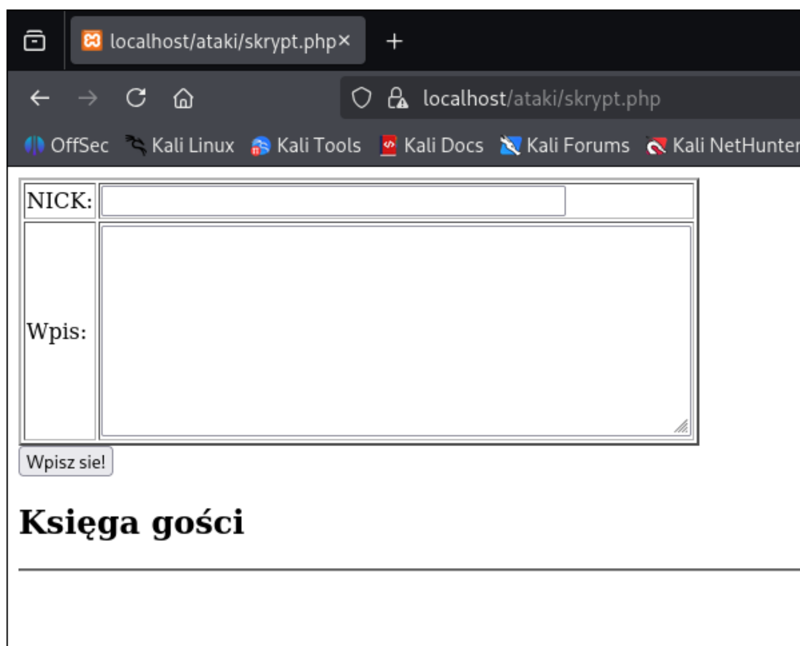
Deprecated: mysqli_real_escape_string(): Passing null to parameter #2 ($string) of type string is deprecated in **/opt/lampp/htdocs/ataki/skrypt2.php** on line **28**
Zapytanie SQL: **SELECT id FROM users WHERE login = 'admin\'-- ' AND pass = ''**

Error (Błędne logowanie)

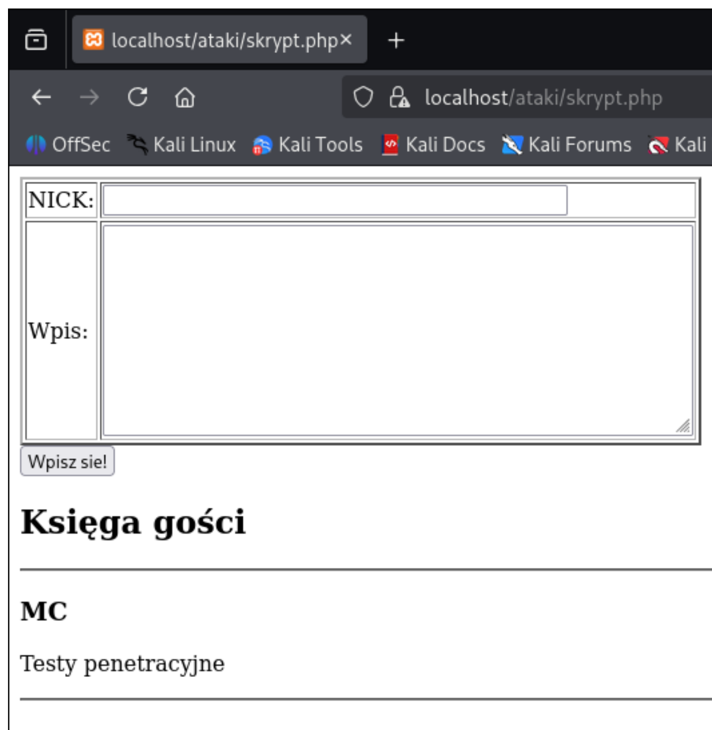
II. XSS – Cross-Site Scripting

Często bagatelizuje się zagrożenie płynące z XSS, uważając je za mało szkodliwe. Aby zrozumieć skalę problemu, przeanalizujemy prosty skrypt księgi gości (`skrypt.php`), który zapisuje wpisy w pliku tekstowym `ksiega.txt`.

1. Aby uruchomić skrypy wpisz w przeglądarce następujący adres: https://localhost/ataki/skrypt.php



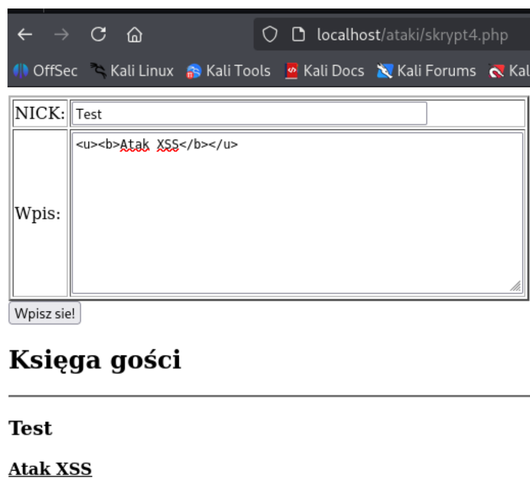
2. Dodaj przykładowe wpisy:



Widzimy, że skrypt działa poprawnie. Pojawia się jednak pytanie: czy jest on bezpieczny?

1. Sprawdźmy to, używając prostych znaczników HTML w treści wpisu:

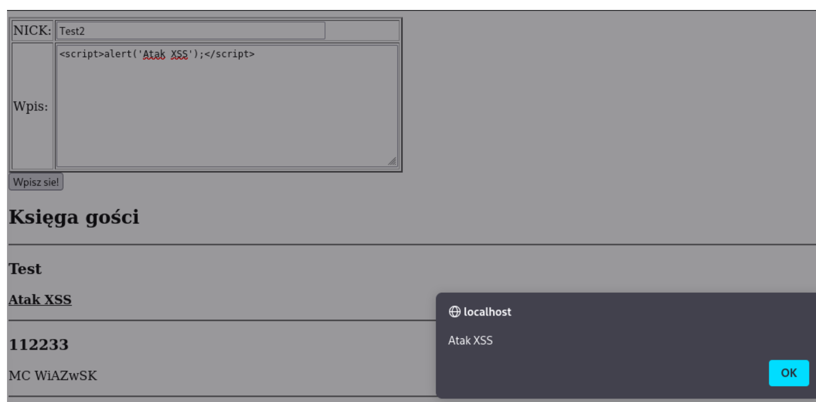
```
<u><b>Atak XSS</b></u>
```



Jeżeli wpis zostanie poprawnie wyświetlony, oznacza to, że formularz nie filtruje danych wejściowych. W takiej sytuacji możliwe jest wstawienie dowolnych znaczników HTML, a nawet kodu JavaScript.

2. Spróbujmy więc wprowadzić prosty skrypt JS:

```
<script>alert('Atak XSS');</script>
```



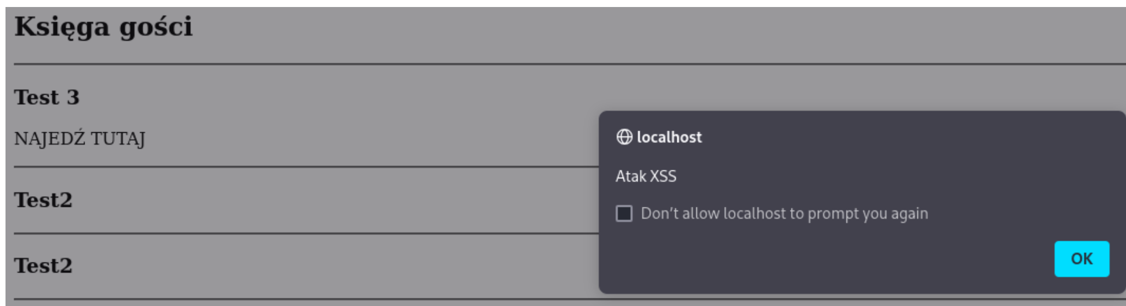
W efekcie, przy każdym odświeżeniu strony, użytkownik zobaczy komunikat alertu – sprawdź.

W prawdziwym ataku zamiast prostego alertu, skrypt mógłby po cichu wysłać ciasteczka sesyjne użytkownika na serwer atakującego.

Webmasterzy często próbują chronić się za pomocą funkcji `strip_tags()`, która usuwa tagi HTML. Czasami jednak dopuszczają niektóre znaczniki, uznając je za bezpieczne (np. <p> dla akapitów).

Atak przez zdarzenia (`Event Handlers`): Nawet "bezpieczny" tag <p> może zostać wykorzystany do ataku, jeśli przyjmuje parametry zdarzeń JavaScript, takie jak `onmouseover`.

3. Wpisz w formularzu: `<p onmouseover="alert('Atak XSS')">NAJEDŹ TUTAJ</p>`



Najechanie na linię utworzonego akapitu spowoduje wyzwolenie alertu JS

III. XSS – Całkowita modyfikacja strony

Jednym z najbardziej widowiskowych efektów ataku XSS jest całkowite zasłonięcie oryginalnej treści strony i wyświetlenie własnego komunikatu, np. słynnego "Hacked By...". Wykorzystuje się do tego znacznik `<div>` z odpowiednio skonstruowanymi stylami CSS.

Atak polega na wstrzyknięciu kodu HTML, który tworzy nową warstwę na samym wierzchu strony. Dzięki pozycjonowaniu absolutnemu (position:absolute) i bardzo wysokiemu indeksowi z-axis (z-index:101), wstrzyknięty element przykrywa wszystkie inne elementy witryny.

1. Kod ataku do wklejenia w formularz księgi gości (zamień Anonim na swój numer indeksu):

```
<div style="text-align:center; color:#FF0000; background:#000000; position:absolute; top:0; left:0; right:0; bottom:0; padding:0; margin:0; height:100000px; z-index:101; padding-top:50px; font-size:50px; font-family:Verdana;">

    Hacked by Anonim

</div>
```