Do realizacji poniższych ćwiczeń stwórz nowy projekt o nazwie np. PHPBezpieczenstwo.

Ćw.1. Niebezpieczne rozszerzenie (.inc, .txt) oraz przechowywanie kluczowych plików.
Należy unikać nadawania plikom dołączanym do PHP rozszerzenia innego niż ***.php***, ponieważ jest możliwe bezpośrednie wyświetlenie kodu źródłowego takiego pliku.
Jest to szczególnie niebezpiecznie, gdy w pliku znajdują się hasła np. do bazy danych.
Utwórz nowy plik ***haslo.txt*** ze skryptem:

```
<?php
      $host='localhost';
      $user='root';
      $pass='toor';
      $dbname='testowa';
      echo 'Test rozszerzenia .txt';
?>
```

Następnie spróbuj wykonać skrypt wpisując w pasku adresowym przeglądarki ścieżkę i nazwę tego pliku:

```
localhost/PHPBezpieczenstwo/hasla.txt
```

Co zostało wyświetlone w oknie przeglądarki?

Utwórz nowy plik ***polaczenie.php***:

```
<?php
      include 'haslo.txt';
?>wyświetlenia
```

Uruchom skrypt *polaczenie.php*.
W efekcie wykonania tego skryptu wykonała się instrukcja *echo* ze skryptu *haslotxt*. Jednak nadal można podejrzeć kod skryptu z hasłem.
Aby to wyeliminować zmień rozszerzenie pliku *haslo.txt* na *haslo.php* oraz popraw rozszerzenie w pliku *polaczenie.php*. Uruchom ponownie oba skrypty.
Jeżeli serwer jest prawidłowo skonfigurowany, to nie wyświetli się już kod pliku, lecz tylko wykona się instrukcja *echo*.

Jednak nadal istnieje możliwość wyświetlenia zawartości pliku.
Żeby zminimalizować to niebezpieczeństwo skopiuj plik *hasla.php* do katalogu znajdującego się powyżej katalogu *htdocs* (zakładamy, że pliki naszej aplikacji umieszczone są na serwerze WWW w podkatalogu znajdującym się w folderze *htdocs*). Popraw także linię w pliku *polaczenie.php*:

```
include 'haslo.php';
```

na:

```
include '../../haslo.php';
```

Wykonaj skrypt *polaczenie.php*, a następnie wpisz do paska adresowego:

```
http://localhost/PHPBezpieczenstwo/../../haslo.php
```

Serwer powinien zwrócić błąd.
Problemem jest to, że nie na wszystkich serwerach możemy wgrać plik do katalogu znajdującego się powyżej, co jest wynikiem konfiguracji serwera, który nie pozwala użytkownikowi na swobodne eksplorowanie katalogów.

Ćw.2. Atak typu Cross-Site Scripting oraz HTML Injection.

Utwórz następujący plik o nazwie *css.php*:

```
<form action="css.php" method="post">
   <textarea name="tekst"></textarea><br />
   <input type="submit" name="wyslij" value="Wyślij" />
</form>
<div>
<?php
  if (filter_input(INPUT_POST,'wyslij'))
                    echo $_POST['tekst'];
?>
</div>
```

Uruchom skrypt i wpisz do pola *tekst* dowolny kod HTML, np. odnośnik do strony:
cs.pollub.pl
a następnie wyślij formularz.
Co się stało po wysłaniu formularza?

Następnie zmodyfikuj skrypt tak, aby znaczniki HTML były zamieniane na postać niewykonywaną, czyli znaki < oraz > zostaną zamienione na encje < oraz >, a znak & w encjach zostanie zamieniony na &.
Zmodyfikuj w tym celu linię z poleceniem *echo*:

```
echo htmlspecialchars($_POST['tekst']);
```

Następnie ponownie przetestuj skrypt. Powinien teraz wyświetlić kod HTML, a nie wykonać go. Zamiast *htmlspecialchars* możesz także przetestować funkcję *strip_tags*, która usuwa wszystkie znaczniki HTML z podanego ciągu.
Czy widać na czym polega różnica w działaniu obu funkcji *htmlspecialchars* i *strip_tags*?
Zamiast tych funkcji można skorzystać z odpowiednich filtrów:
/FILTER_SANITIZE_FULL_SPECIAL_CHARS i FILTER_SANITIZE_STRING

Ćw.3. Atak typu Directory Traversal

Utwórz skrypt *DirTrav.php*:

```
<?php
function download($patch = '.')
{
  $katalog = @dir($patch) or die ('Brak dostępu do katalogu.');
  while ($plik_kat = $katalog->read())
   if(is_file($patch.'/'.$plik_kat))
   echo '<a href="'.$patch.'/'.$plik_kat.'">'.$plik_kat.'</a><br />';
   elseif (is_dir($patch.'/'.$plik_kat))
    {
      echo '<a href="DirTrav.php?patch='.$patch.'/'.$plik_kat.'">'.$plik_kat.'</a>
                    <br />';
    }
   $katalog->close();
}
if (!isset($_GET['patch']))
  download();
else
  download($_GET['patch']);
?>
```

Uruchom skrypt. Wyświetli się lista plików i katalogów z poziomu katalogu, w którym znajduje się skrypt. Spróbuj przejść do folderu nadrzędnego klikając *".."*.
Czy możemy dowolnie poruszać się po strukturze katalogów serwera Apache?

Zabezpiecz skrypt zmieniając linię:
```
download($_GET['patch']);
```
na
```
download(str_replace('..','',$_GET['patch']));
```

Przetestuj teraz próbę cofnięcia się do katalogu nadrzędnego. Powinna zakończyć się ona wyświetleniem zawartości katalogu zawierającego skrypt.

Ćw.4. Atak typu File Inclusion.
Jest to jeden z najpoważniejszych ataków, ponieważ atakujący może uruchomić dowolny skrypt PHP. Może na przykład uzyskać informacje na temat naszej aplikacji, podejrzeć kod skryptów PHP czy hasła do bazy danych (pomimo zabezpieczeń wprowadzonych w **Ćw.1)**.
W tym ćwiczeniu złośliwy skrypt będzie jedynie wyświetlał tekst, ale w rzeczywistym ataku może to być skrypt, którego zadaniem będzie uzyskanie informacji o aplikacji.

Utwórz plik *inclusion.php*:
```
<?php
  if (isset($_GET['plik']))
    include($_GET['plik'].'.txt');
  else
    echo 'Nie podano pliku!';
?>
```
Utwórz także plik tekstowy *skrypt.txt* i umieść go poza katalogiem WWW, najlepiej w *htdocs*:
```
<?php
     echo 'Strona nie jest zabezpieczona';
?>
```
Wpisz do paska adresowego następujący adres:
```
localhost/PhpBezpieczenstwo/inclusion.php?plik=http://localhost/skrypt
```

Wykonał się kod PHP z pliku *skrypt.txt* lub pojawił się błąd spowodowany konfiguracją PHP.
Żeby zobaczyć efekt ataku należy w pliku *php.ini* zmodyfikować:
```
allow_url_include = Off
```
na:
```
allow_url_include = On
```

Po tej modyfikacji zrestartuj serwer Apache i ponownie przetestuj skrypt Tym razem próba ataku się powiedzie).

Następnie zabezpiecz skrypt modyfikując:
```
if (isset($_GET['plik']))
  include($_GET['plik'].'.txt');
```
na:
```
if (isset($_GET['plik']))
{
  $plik=str_replace('\\','/',$_GET['plik']);
  $plik=str_replace('.','/',$plik);
  $filearr=explode('/',$plik);
  $plik=$filearr[count($filearr)-1];
  include($plik.'.txt');
}
```

Uruchom zmodyfikowany skrypt i spróbuj ponownie wykonać atak .
Pojawi się błąd o braku możliwości dołączenia pliku (jeśli tylko *skrypt.txt* znajduje się w innym katalogu niż *inclusion.php).*
Po zakończeniu zadania przywróć ustawienia w *php.ini* na:

```
allow_url_include = Off.
```

Ćw.5. Atak typu SQL Injection.
Jest to jeden z najpoważniejszych ataków na aplikacje internetowe. Atakujący może np. wykasować rekordy lub tabele. zmienić i wstawiać nowe rekordy, co może prowadzić do przejęcia kontroli nad stroną.
Jeśli atakujący zdoła dodać użytkownika do bazy z prawami administratora lub będąc zwykłym użytkownikiem zmieni uprawnienia na administratora, to będzie mógł zarządzać naszym systemem.

Przed wykonaniem ćwiczenia sprawdź, czy opcja *magic_quotes_gpc* w pliku *php.ini* jest ustawiona na *Off*. Jeżeli jest *On*, to wyłącz ją na czas tego ćwiczenia.

Uruchom *PHPMyAdmin* i sprawdź czy jest tam baza o nazwie *testowa*. Jeżeli nie ma to utwórz ją. Następnie dla tej bazy wykonaj zapytania:

```
CREATE TABLE `testowa`.`strony` (`id` INT NOT NULL AUTO_INCREMENT PRIMARY KEY
,`tytul` VARCHAR( 30 ) NOT NULL) ENGINE = MYISAM ;
INSERT INTO `testowa`.`strony` (`id` ,`tytul`) VALUES (NULL , 'A'), (NULL , 'B'),
(NULL , 'C' ), (NULL , 'D');
```

Utwórz podatny na atak skrypt o nazwie *sql.php*, który będzie wyświetlał tytuły z bazy (lub jeden z nich po podaniu parametru *id* przekazanego metodą *POST)* – skorzystaj z klasy *Baza* z poprzednich laboratoriów:

```
<?php
 include_once 'lib/Baza.php';
 $ob = new Baza('localhost','root','','testowa');
 if (isset($_POST['id']))
 {
   echo 'Wybrany:<br />';
   $id=$_POST['id'];
   echo 'SQL: SELECT tytul FROM strony WHERE id="'.$id.'";<br />';
   echo $ob->select('SELECT id,tytul FROM strony WHERE
                       id="'.$id.'";',array('id','tytul'));
}
else
{
  echo '<form action="sql.php" method="post">';
  echo 'Wpisz numer ID do pokazania: <input type="text" name="id">';
  echo '<input type="submit" value="Uruchom"><br />';
  echo 'Wszystkie:<br />';
  echo $ob->select('SELECT id,tytul FROM strony;',array('id','tytul'));
}
?>
```

Przetestuj działanie skryptu – czy wyświetla poprawnie wszystkie rekordy i czy po podaniu **id** w polu tekstowym wyświetla właściwy rekord?
Spróbuj dokonać ataku *SQL Injection*, który spowoduje wyświetlenie dodatkowego rekordu o *id=2*. Wpisz do pola tekstowego następujący tekst:

```
1" or id="2
```

Spróbuj wykonać teraz atak wpisując do pola tekstowego:

```
1";DELETE FROM strony WHERE id="1
```

Prawdopodobnie nie uda się wykonać ataku z dodatkowym zapytaniem (np. kasującego rekordy), ponieważ funkcje PHP uniemożliwiają wykonanie większej liczby zapytań w jednej metodzie *query()*.

Chcąc zastosować wiele zapytań w jednej funkcji, należałoby użyć metody obiektu *mysqli*: *multi_query()* zamiast *query()*.

W celu zabezpieczenia skryptu przed atakami SQL Injection należy wykonać następującą modyfikację:
Linię:

```
$id=$_POST['id'];
```

zamienić na:

```
$id=addslashes($_POST['id']);
```

Przetestuj ponownie wyświetlanie dwóch rekordów (nie powinno już zadziałać).

Możesz dodać nową metodę do klasy *Baza*, która będzie zabezpieczała wprowadzone dane przez użycie dedykowanej metody z klasy *mysqli*:

```
public function protect_string($str) {
   return $this->mysqli->real_escape_string($str);
}
```

Możesz również przygotować metodę dla danych, które powinny mieć wartości całkowite:

```
public function protect_int($nmb) {
                return (int)$nmb;
}
```

Zmień zmodyfikowaną wcześniej linię z pliku *sql.php* na:

```
$id=$ob->protect_string($_POST['id']);
```

Przetestuj ponownie atak. Możesz także użyć drugiej metody, czyli *protect_int*, ponieważ pole *id* jest typu INT. Atak w obu przypadkach się już nie powiedzie.

Korzystając z biblioteki PDO zamiast *query()*, można użyć tzw. podpinania, które opisane jest na stronie:

```
https://pl.wikibooks.org/wiki/PHP/Biblioteka_PDO#Podpinanie
```

Ćw.6. Bezpieczeństwo sesji w PHP

Mechanizm sesji w PHP nie jest doskonały i należy zadbać również o jego bezpieczeństwo. Wadą tego systemu jest zastosowanie identyfikatora w adresie URL. Przez odpowiednie ustawienia w pliku *php.ini* można wymusić stosowanie wyłącznie ciasteczek - za pomocą opcji *session.use_only_cookies* oraz *session.trans_sid*.
Sesje mają domyślnie dość długi czas trwania, co daje atakującemu dużo czasu na przygotowanie ataku. Należy więc samodzielnie kontrolować ten czas.
W ćwiczeniu przedstawione zostanie zabezpieczenie przed atakiem przejęcia identyfikatora sesji oraz wymuszenia sesji.

Przygotuj skrypt *sesja.php:*

```
<?php
    session_start();
    echo 'ID: '.session_id().'<br />';
    if (isset($_GET['PHPSESSID']))
    echo 'PHPSESSID: '.$_GET['PHPSESSID'].'<br />';
?>
```

Po uruchomieniu skryptu pojawi się wygenerowany identyfikator. Uruchom ponownie przeglądarkę, wpisz w adresie:

```
sesja.php?PHPSESSID=
```

i po znaku równości wprowadź swój własny identyfikator oraz załaduj stronę.
Zauważ, że identyfikator został ustawiony na ten wpisany ręcznie. Taki sposób daje atakującemu możliwość przechwycenia identyfikatora zalogowanego użytkownika, a więc uzyskania jego uprawnień.
Można też podesłać niezalogowanemu użytkownikowi taki adres a po zalogowaniu się tego użytkownika - przechwycić sesję.
Zabezpieczenie przed taką próbą przejęcia sesji polega na odpowiednich ustawieniach w pliku *php.ini*, ale należy również regenerować ID sesji.
Zmodyfikuj skrypt – dodaj po *session_start():*

```
if (!isset($_SESSION['our_own']))
{
     session_regenerate_id();
     $_SESSION['our_own'] = true;
}
```

Przetestuj teraz próbę ustawienia PHPSESSID – skrypt nie będzie już reagował na wprowadzony ręcznie identyfikator.

Ćw.7. Atak typu Cross-Site Request Forgery

Atak CSRF polega na tym, że atakujący umieszcza adres do pewnego skryptu jako adres obrazka lub ramki. Skrypt ten najczęściej jest zabezpieczony przed wykonaniem dla użytkowników nieupoważnionych. Ale jeżeli obrazek lub ramka zostanie otwarta przez zalogowanego użytkownika z odpowiednimi uprawnieniami w systemie (np. administratora), to ten użytkownik może nieświadomie wykonać niebezpieczną akcję. Atak jest szczególnie niebezpieczny, jeśli parametry są przekazywane metodą GET.
Wykonanie tego ataku można uniemożliwić poprzez wysyłanie kluczowych parametrów metodą POST. Nie zawsze jest to skuteczne, ponieważ można wysyłać żądania metodą POST przy użyciu JavaScript. Dlatego nie należy odwiedzać innych stron, jeśli jesteśmy zalogowani w jakimś serwisie.
W swoich skryptach należy zabezpieczyć się przed próbą osadzenia JavaScript.
Skrypt w ćwiczeniu będzie używał poprzednio utworzonej tabeli w bazie danych.

Przygotuj skrypt *delete.php*:

```
<?php
  include_once 'lib/Baza.php';
  $ob = new Baza('localhost','root','','testowa');
  if (isset($_GET['id']))
  if ($ob->delete('DELETE FROM strony WHERE id="'.
                      $ob->protect_int($_GET['id']).'"'))
    echo 'Skasowano rekord o id='.$_GET['id'];
else
    echo 'Nie można skasować rekordu!';
?>
```

Przygotuj także *plik csrf.html* i umieść w nim:

```
<img src="delete.php?id=1" alt="Delete" />
```

Możesz oczywiście podać inny istniejący identyfikator.
Uruchom plik *csrf.html* i sprawdź przy użyciu *PHPMyAdmina*, czy rekord o wskazanym *id* został usunięty.
W celu zabezpieczenia skryptu – w pliku *delete.php* zmień wszystkie odwołania GET na POST. Jeszcze raz wywołaj plik *csrf.html* – zmień wcześniej identyfikator w obrazku lub przywróć skasowany rekord.
Do pliku *csrf.html* dodaj formularz do kasowania rekordu i sprawdzić jego działanie:

```
<form action="delete.php" method="post">
      ID:<input type="number" name="id" />
      <input type="submit" value="Kasuj" />
</form>
```

Formularz symuluje przekazanie parametrów metodą POST, które można zrealizować przy pomocy JavaScript. Wtedy też możemy wykorzystać zwykłe odnośniki, które zamiast prowadzić do pliku *delete.php* prowadzą do funkcji JS, która wyśle żądanie HTTP metodą POST.

Ćw.8. Niebezpieczne funkcje PHP
Jedną z takich funkcji jest *system(),* która pozwala na wykonywanie poleceń systemowych przez skrypt PHP. Nie należy do niej przekazywać parametrów, podawanych przez użytkownika. W przypadku takiej konieczności należy przefiltrować parametry za pomocą funkcji *escapeshellcmd().* Funkcja usuwa znaki pozwalające na wykonywanie innych komend.
Kolejną niebezpieczną funkcją jest exec().
Wiele serwerów blokuje możliwość wykorzystywania tych funkcji za pomocą opcji w pliku *php.ini*:

```
disable_functions = system,exec
```

Aby przetestować funkcję *system(),* przygotuj skrypt *system.php*:

```
<?php
     system('cmd.exe /C '.$_GET['cmd']);
?>
```

Uruchom skrypt i wpisz w adresie:

```
system.php?cmd=ping localhost
```

W ten sposób możemy wykonać dowolną komendę. Na systemach Uniksowych możemy wstawić średnik i wykonać wiele komend.
Zmodyfikuj skrypt:

```
system('cmd.exe /C '.escapeshellcmd($_GET['cmd']));
```

Nie zabezpieczy to skryptu przed wykonaniem powyższego przykładu w systemie Windows. Dlatego lepiej **nigdy** nie należy używać tych funkcji.