Politechnika Lubelska

PODRĘCZNIKI

Grzegorz Kozieł, Piotr Muryjas

Bezpieczne aplikacje internetowe w PHP

```php
<?php

class Crypto
{
    public function __construct()
    {
    }

    function calculate_bcrypt($password): string
    {
        $options = [
            'cost' => 12
        ];
        return password_hash($password, algo: PASSWORD_BCRYPT, $options);
    }

    function  verify_bcrypt($password,$hash): bool
    {
        if (password_verify($password, $hash)) {
            return true;
        } else {
            return false;
        }
    }
}
```

Lublin 2022

Bezpieczne aplikacje internetowe w PHP

Podręczniki – Politechnika Lubelska

POLITECHNIKA
LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI
I INFORMATYKI

Grzegorz Kozieł, Piotr Muryjas

Bezpieczne aplikacje internetowe w PHP

POLITECHNIKA
LUBELSKA
WYDAWNICTWO

Lublin 2022

Recenzenci:
dr. hab. Bogdan Księżopolski, PJATK
dr Rafał Stęgierski, Politechnika Lubelska

,,Konkurs na wydanie podręcznika akademickiego lub skryptu" edycja I

Publikacja wydana za zgodą Rektora Politechniki Lubelskiej

ISBN: 978-83-7947-538-4

Wydawca: Wydawnictwo Politechniki Lubelskiej
www.wpl.pollub.pl
ul. Nadbystrzycka 36C, 20-618 Lublin
tel. (81) 538-46-59

Druk: Drukarnia Akapit Sp. z o. o.
www.drukarniaakapit.pl

Nakład: 50 egz.

Spis treści

Streszczenie

Aplikacje internetowe są nieodłączną częścią naszego życia. Znakomita większość tworzonych obecnie systemów udostępniana jest w postaci aplikacji internetowych. Pozwala to na łatwy dostęp do nich z każdego urządzenia posiadającego łącze internetowe oraz przeglądarkę. Powszechność i łatwość dostępu stanowi niewątpliwą zaletę, ale też zagrożenie. Każdy może podjąć próbę ataku. Dlatego też istotne jest zabezpieczenie aplikacji we właściwy sposób.

W niniejszym opracowaniu przedstawione zostały podstawowe techniki zabezpieczania aplikacji internetowych. Czytelnik znajdzie tu zarówno omówienie technik ataku, jak i sposobów ochrony przed nimi. Wszystkie omawiane zagadnienia zawierają przykłady praktyczne w postaci kodu PHP.

Niniejsza książka przeznaczona jest dla czytelników, którzy posiadają podstawową znajomość języków: PHP, HTML, CSS, SQL oraz relacyjnej bazy danych MySQL. Wszystkie przykłady zawarte w książce będą prezentowane w języku PHP.

Wszystkie prezentowane treści oraz kody programów zostały przez autorów przetestowane. Środowiskiem, jakie zostało użyte podczas opracowywania niniejszego opracowania było środowisko PhpStorm 2021.3.1. Użyto również silnika baz danych MySQL dostępnego w pakiecie XAMPP pracującego pod kontrolą systemu Linux Kubuntu 20.04 LTS.

Słowa kluczowe: aplikacje internetowe, PHP, cyberbezpieczeństwo

Abstract

Web applications are a part of our life. The majority of modern systems are created as web applications. It enables users to get access to them with any device that has an Internet connection and a browser. Easy access and universality are great advantages but they are also threats. Everybody can try to attack an internet application. Therefore, it is important to secure the application in the right way.

This paper presents the basic techniques of securing Internet applications. The reader will find here both a discussion of attack techniques and ways to protect against them. All discussed issues include practical examples in the form of PHP code.

This book is intended for readers who have a basic knowledge of PHP, HTML, CSS, SQL, and the MySQL relational database. All examples contained in the book will be presented in PHP.

All presented content and program codes have been tested by the authors. The environment that was used during the development of this study was the PhpStorm 2021.3.1 environment. The MySQL database engine available in the XAMPP package running under Linux Kubuntu 20.04 LTS was also used.

Keywords: web applications, PHP, cybersecurity

1. Wprowadzenie

Język PHP pozwala na tworzenie aplikacji internetowych, które korzystają z danych przechowywanych w bazach danych [8]. W niniejszym rozdziale omówione zostaną sposoby nawiązywania połączenia z bazą danych oraz wykonywania podstawowych poleceń języka SQL[8]. Utworzymy również zrąb aplikacji, w której będziemy implementować zabezpieczenia – wykorzystamy ją w dalszej części książki. Czytelnicy, którzy znają tę tematykę i potrafią samodzielnie stworzyć aplikację w języku PHP mogą przejść od razu do rozdziału drugiego. Kod aplikacji wykonanej w rozdziale pierwszym dostępny jest pod adresem https://drive.google.com/drive/folders/1dwGvfRnWPseFJFezUn8qa4YqqH1Cyg5B?usp=sharing.

1.1. Obsługa bazy danych

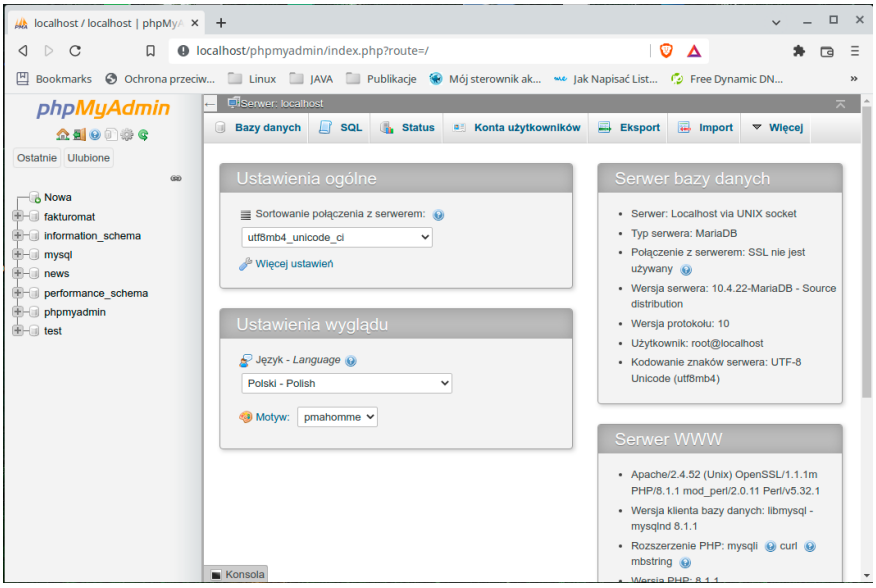
Zanim będziemy mogli rozpocząć pracę z bazą danych, musimy ją utworzyć. Wymagane jest więc posiadanie silnika baz danych. W niniejszej książce wykorzystywać będziemy oprogramowanie XAMPP. Można je pobrać ze strony https://www.apachefriends.org/pl/index.html.

Po zainstalowaniu przechodzimy do katalogu, w którym zainstalowane zostało oprogramowanie i uruchamiamy je za pomocą polecenia `sudo ./lampp start`, tak jak zaprezentowano to na rysunku 1.1. Spowoduje to uruchomienie wszystkich usług wchodzących w skład pakietu, w tym niezbędnych dla nas usług serwera WWW oraz serwera baz danych.

```
grzes@LAP:/opt/lampp$ sudo ./lampp start
Starting XAMPP for Linux 8.1.1-2...
XAMPP: Starting Apache...ok.
XAMPP: Starting MySQL...ok.
XAMPP: Starting ProFTPD...ok.
```

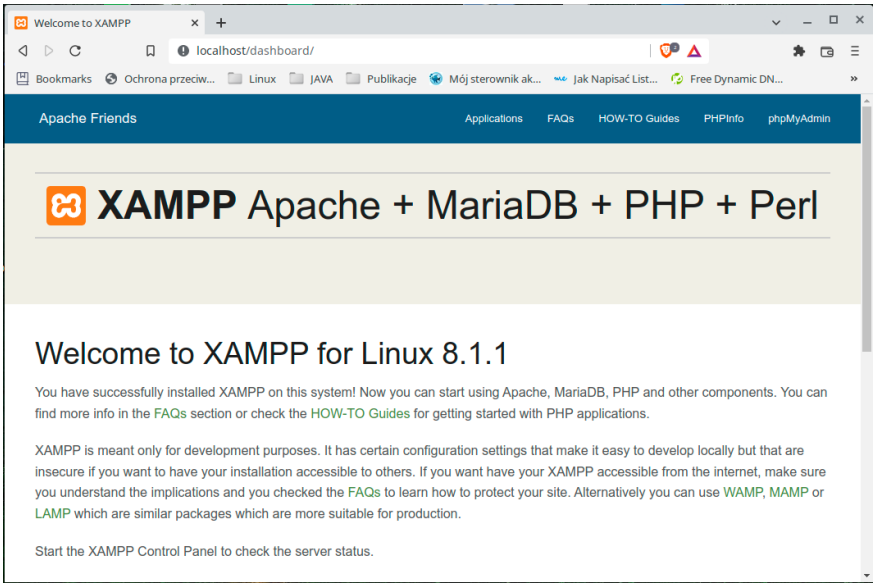
Rys.1.1. Uruchomienie pakietu XAMPP

W tym momencie możemy sprawdzić czy uruchomione serwery działają poprawnie. W przeglądarce wpisujemy adres „localhost" i patrzymy, czy otworzyła się strona gówna pakietu XAMPP. Powinniśmy uzyskać efekt taki. jak widnieje na rysunku 1.2.



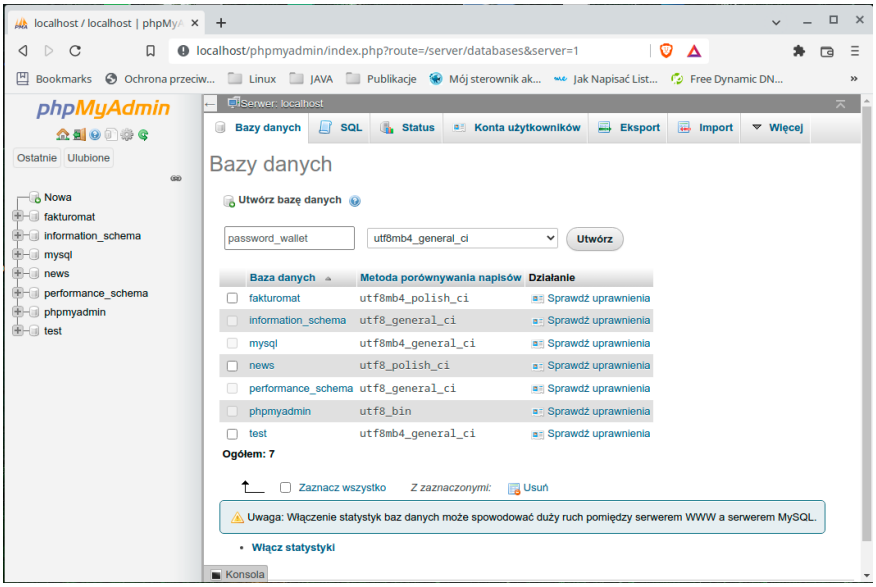
Rys.1.2. Strona główna pakietu XAMPP

Teraz możemy przejść do tworzenia bazy danych. W tym celu klikamy na łącze *phpMyAdmin* widoczne w prawym górnym rogu strony. Przeniesie nas to do strony stanowiącej interfejs usługi phpMyAdmin służącej do zarządzania bazą danych. Wygląd tej strony zaprezentowany został na rysunku 1.3.



Rys.1.3. Strona narzędzia phpMyAdmin

Korzystając z łącza *Nowa* widniejącego w panelu po lewej stronie ekranu, przechodzimy do widoku tworzenia nowej bazy danych. Tam podajemy nazwę bazy danych i klikamy na przycisk *Utwórz*. Spowoduje to utworzenie nowej bazy danych. Będziemy ją wykorzystywać w dalszej części podręcznika. Widok tworzenia bazy danych przedstawiono na rysunku 1.4.



Rys.1.4. Widok tworzenia bazy danych

Utworzona baza danych pojawi się na liście baz widocznej w lewym panelu. Wystarczy kliknąć jej nazwę by ją wybrać i rozpocząć z nią pracę. Aktualnie nasza baza jest pusta. Musimy w niej stworzyć strukturę pozwalającą na przechowywanie danych naszej aplikacji.

Do celów prezentacji technik zabezpieczeń utworzymy aplikację portfela haseł. Będzie to aplikacja umożliwiająca przechowywanie haseł użytkownika w bezpieczny sposób. Stwórzmy więc prostą bazę danych. Będzie ona zawierała dwie tabele:

- *users* – służącą do przechowywania informacji o użytkowniku oraz jego danych dostępowych do aplikacji,
- *passwords* – przeznaczoną do przechowywania haseł użytkowników.

Do utworzenia bazy użyjemy skryptu zaprezentowanego na listingu 1.1.

Listing 1.1. Skrypt tworzący bazę danych

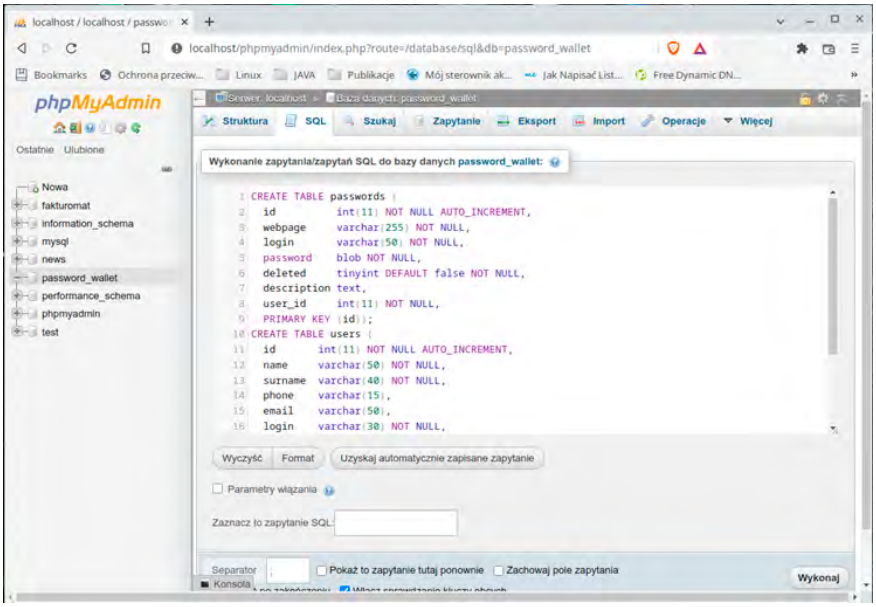
```
CREATE TABLE passwords (
  id          int(11) NOT NULL AUTO_INCREMENT,
  webpage     varchar(255) NOT NULL,
  login       varchar(50) NOT NULL,
  password    blob NOT NULL,
  deleted     tinyint DEFAULT false NOT NULL,
  description text,
  user_id     int(11) NOT NULL,
  PRIMARY KEY (id));
CREATE TABLE users (
  id       int(11) NOT NULL AUTO_INCREMENT,
  name     varchar(50) NOT NULL,
  surname  varchar(40) NOT NULL,
  phone    varchar(15),
  email    varchar(50),
  login    varchar(30) NOT NULL,
  password blob NOT NULL,
  salt     blob,
  deleted  tinyint(1) DEFAULT false NOT NULL,
  PRIMARY KEY (id));
ALTER TABLE passwords ADD CONSTRAINT FKpasswords699033
FOREIGN KEY (user_id) REFERENCES users (id) ON UPDATE
Cascade ON DELETE Cascade;
```

Skrypt widoczny na listingu 1.1 możemy wkleić do okna wykonywania zapytań narzędzia phpMyAdmin. Wybieramy więc naszą bazę z listy baz. Następnie przechodzimy na zakładkę SQL, gdzie wklejamy kod naszego skryptu. Klikamy na przycisk *Wykonaj*, tak jak to zaprezentowano na rysunku 1.5. Po pomyślnym wykonaniu operacji powinniśmy zobaczyć ekran widoczny na rysunku 1.6.

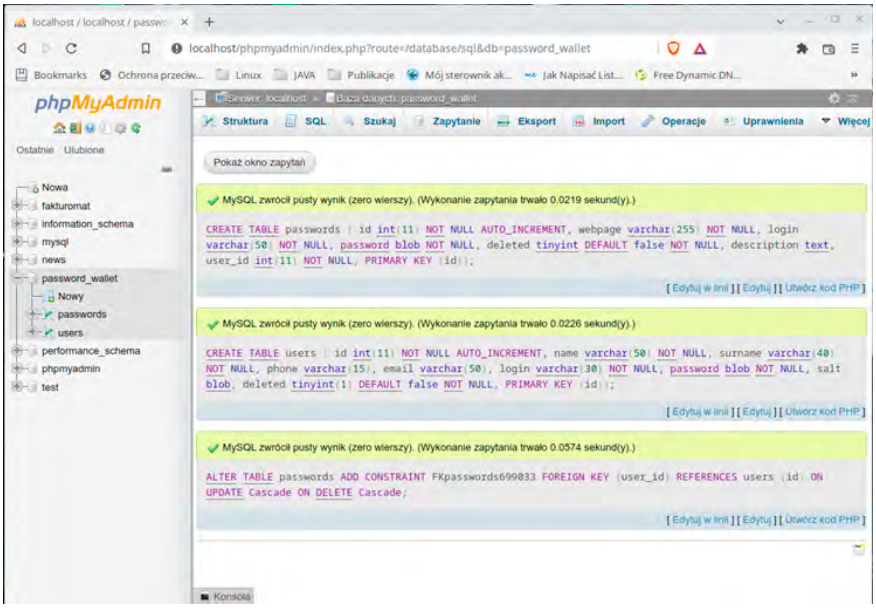
Wypełnijmy jeszcze naszą bazę danymi testowymi. W tym celu, w oknie wykonywania zapytań wprowadzamy skrypt z listingu 1.2. Spowoduje on utworzenie dwóch rekordów w tabeli users i wypełni wybrane pola danymi. W tym momencie są to jedynie dane przykładowe, które mają za zadanie pomóc nam zweryfikować czy aplikacja, którą będziemy tworzyć poprawnie współpracuje z bazą danych. Jeśli wszystko wykonamy poprawnie, powinniśmy zobaczyć potwierdzenie, jakie zaprezentowano na rysunku 1.7.

Listing 1.2. Skrypt wstawiający rekordy do tabeli users

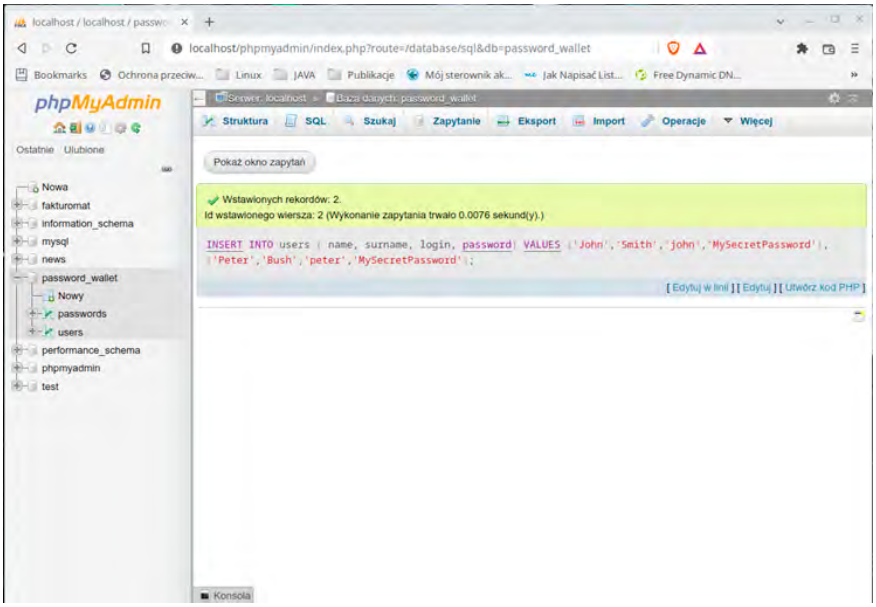
```
INSERT INTO users ( name, surname, login, password)
VALUES
  ('John','Smith','john','MySecretPassword'),
  ('Peter','Bush','peter','MySecretPassword');
```



Rys.1.5. Widok formularza wykonania zapytań



Rys.1.6. Widok strony potwierdzającej wykonanie instrukcji SQL



Rys.1.7. Widok *strony potwierdzającej wstawienie rekordów do tabeli users*

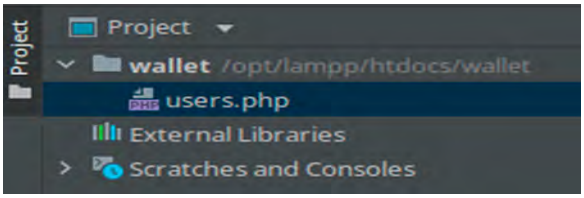
1.2. Nawiązanie połączenia z bazą danych

W niniejszym rozdziale wykonamy prostą aplikację współpracującą z bazą danych. Będzie ona pozwalała na wyświetlanie listy użytkowników i dodawanie nowych. Aplikacja ta będzie pozbawiona zabezpieczeń.

Rozpoczynamy od utworzenia projektu. W nim tworzymy plik users.php, w którym umieścimy kod strony odpowiedzialnej za wyświetlenie listy użytkowników. Katalog aplikacji musi znajdować się wewnątrz katalogu htdocs pakietu XAMPP. Tylko wówczas nasz serwer WWW będzie mógł udostępniać tworzoną aplikację. W tym podręczniku użyto katalogu o nazwie *wallet*. Struktura projektu powinna wyglądać tak, jak to zaprezentowano na rysunku 1.8.

Tworząc kod obsługujący bazę danych posłużymy się tutorialem dostępnym na stronie https://www.w3schools.com/php/default.asp. Zawiera on zarówno objaśnienia, jak i gotowe fragmenty kodu, które możemy wykorzystać.

Tworzenie aplikacji rozpoczniemy od nawiązania połączenia z bazą danych. Użyjemy kodu z listingu 1.3. Wykorzystujemy w nim bibliotekę MySQLi, której przekazujemy dane konfiguracyjne zawierające adres serwera oraz dane użytkownika bazodanowego – jego login i hasło. Skorzystamy z domyślnego konta *root*, które nie posiada hasła. Należy pamiętać, że dostęp do tego konta jest możliwy jedynie z tej samej maszyny, na której działa silnik baz danych.

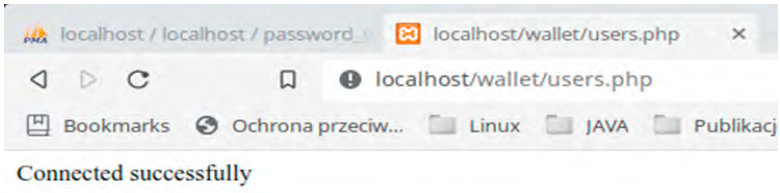


Rys.1.8. Struktura projektu

Listing 1.3. Nawiązanie połączenia z bazą danych

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "password_wallet";
$conn = new mysqli($servername, $username, $password,
$dbname); // Nawiązanie połączenia
if ($conn->connect_error) { // Weryfikacja połączenia
    die("Connection failed: " . $conn->connect_error);
}
echo „Connected successfully";
```

Weryfikujemy, czy połączenie z bazą danych zostało nawiązane. W przeglądarce komputera, na którym działa serwer WWW wpisujemy adres http://localhost/wallet/users.php. Powinniśmy zobaczyć taki efekt, jak zaprezentowano na rysunku 1.9.



Rys.1.9. Wygląd strony users.php

1.3. Pobranie danych z bazy

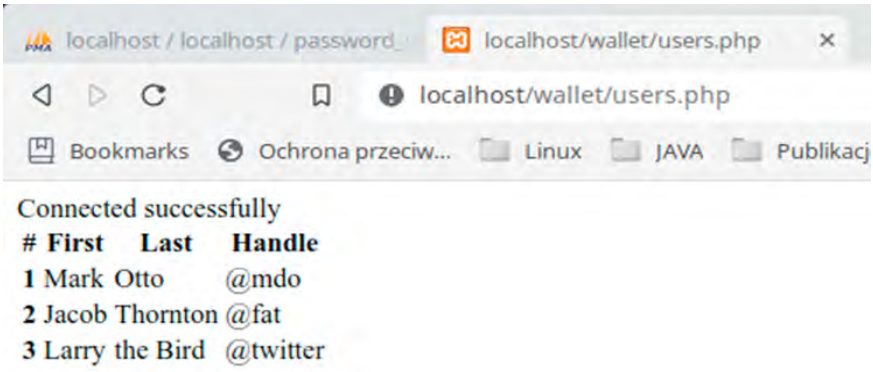
W tym rozdziale zajmiemy się wyświetleniem listy użytkowników zapisanych w bazie danych. Wykonamy też prosty mechanizm filtrowania użytkowników. Listę użytkowników będziemy wyświetlać w tabeli. Zacznijmy więc od jej utworzenia.

Tu również postaramy się przyspieszyć pracę i wykorzystać gotowe szablony. Są one dostępne na wielu stronach internetowych. My skorzystamy z tych, które są dostępne w projekcie bootstrap. Użyjemy również dostępnego tam arkusza CSS.

Lista użytkowników będzie wyświetlana w tabeli. Rozpoczniemy od pobrania szablonu tabeli. Wpisujemy w wyszukiwarce hasło *bootstrap table* i korzystamy z wyników wyszukiwania w celu szybkiego przejścia na stronę z przykładami tabel. Kopiujemy z niej kod wybranej tabeli. Autorzy skorzystali z kodu zaprezentowanego na listingu 1.4. Został on wklejony na końcu pliku users.php. W wyniku tego uzyskaliśmy wygląd strony zaprezentowany na rysunku 1.10.

Listing 1.4. Kod tabeli [źródło: https://getbootstrap.com/docs/4.0/content/tables/]

```
<table class=”table”>
  <thead>
    <tr>
      <th scope=”col”>#</th>
      <th scope=”col”>First</th>
      <th scope=”col”>Last</th>
      <th scope=”col”>Handle</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th scope=”row”>1</th>
      <td>Mark</td>
      <td>Otto</td>
      <td>@mdo</td>
    </tr>
    <tr>
      <th scope=”row”>2</th>
      <td>Jacob</td>
      <td>Thornton</td>
      <td>@fat</td>
    </tr>
    <tr>
      <th scope=”row”>3</th>
      <td>Larry</td>
      <td>the Bird</td>
      <td>@twitter</td>
    </tr>
  </tbody>
</table>
```



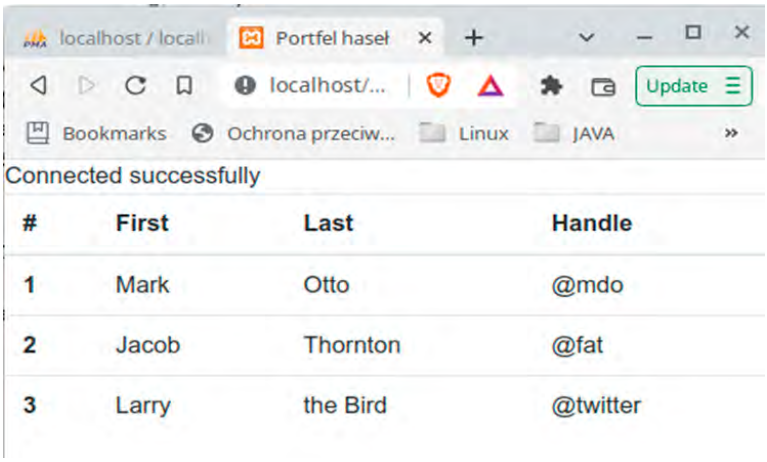
Rys.1.10. Wygląd strony users.php

Tabela pojawiła się na stronie. Jednak jej wygląd odbiega od tego, co oferuje bootstrap. Dołączmy podstawowy arkusz stylów udostępniany przez bootstrap, aby móc skorzystać z zawartych tam stylów. Ze strony https://getbootstrap.com/ pobieramy bibliotekę bootstrap w postaci skompresowanego pliku. Rozkompresowujemy i szukamy wewnątrz pliku o nazwie *bootstrap.min.css*. Plik ten umieszczamy w katalogu css, który tworzymy w katalogu głównym naszej aplikacji. Następnie dołączamy go, jako zewnętrzny arkusz stylów do pliku users.php. Warto jeszcze zadbać o poprawną składnię HTML całego pliku, dodając wymagane znaczniki. Po ich dodaniu kod strony powinien wyglądać tak, jak przedstawiono na listingu 1.5. Wygląd uzyskanej strony powinien być tożsamy z przedstawionym na rysunku 1.11.

Listing 1.5. Kod strony users.php

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
// Create connection
$conn = new mysqli($servername, $username, $password);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully";
?>
<!DOCTYPE HTML>
<html lang="pl">
 <head>
```

```
  <meta http-equiv="Content-Type" content="text/html;
    charset=utf-8" />
  <meta http-equiv="Content-Language" content="pl" />
  <meta name="description" content="" />
  <meta name="keywords" content="" />
  <link rel="stylesheet" href="css/bootstrap.min.css">
  <title>Portfel haseł</title>
 </head>
 <body>
  <header></header>
  <section>
   <table class="table">
    <thead>
     <tr>
      <th scope="col">#</th>
      <th scope="col">First</th>
      <th scope="col">Last</th>
      <th scope="col">Handle</th>
    </tr>
   </thead>
   <tbody>
    <tr>
      <th scope="row">1</th>
      <td>Mark</td>
      <td>Otto</td>
      <td>@mdo</td>
    </tr>
    <tr>
      <th scope="row">2</th>
      <td>Jacob</td>
      <td>Thornton</td>
      <td>@fat</td>
    </tr>
    <tr>
      <th scope="row">3</th>
      <td>Larry</td>
      <td>the Bird</td>
      <td>@twitter</td>
    </tr>
   </tbody>
  </table>
 </section>
 <footer></footer>
 </body>
</html>
```



Rys.1.11. Wygląd strony users.php

Pora teraz wypełnić tabelę danymi pobranymi z bazy danych. Mechanizm i sposób pobierania danych zaprezentowany jest na stronie https://www.w3schools.com/php/php_mysql_select.asp. Skorzystamy z niego. Pobieramy udostępniony kod i za jego pomocą dane pobrane z bazy wyświetlamy w tabeli, którą wcześniej umieściliśmy na stronie. Kod wyświetlający tabelę z danymi został zaprezentowany na listingu 1.6.

Listing 1.6. Kod wyświetlający tabelę z danymi użytkowników

```
<section>

<?php
 $sql = "SELECT * FROM users";
 $result = $conn->query($sql);

 if ($result->num_rows > 0) {
?>
  <table class="table">
   <thead>
    <tr>
     <th scope="col">Name</th>
     <th scope="col">Surname</th>
     <th scope="col">Phone</th>
    </tr>
   </thead>
   <tbody>
    <?php
```

```
    // wyświetl dane poszczególnych wierszy
    while($row = $result->fetch_assoc()) {
    ?>
     <tr>
      <th scope="row"><?php echo $row["id"] ?></th>
       <td><?php echo $row["name"] ?></td>
       <td><?php echo $row["surname"] ?></td>
       <td><?php echo $row["phone"] ?></td>
      </tr>
   <?php
     }
    }
    $conn->close();
   ?>
    </tbody>
  </table>
</section>
```

Nasza aplikacja wyświetla już dane pobierane z bazy danych. Dodamy do niej funkcję filtrowania danych. Chcemy, aby nasza aplikacja posiadała formularz, w którym użytkownik będzie mógł podać ciąg znaków, jaki mają zawierać nazwiska filtrowanych użytkowników. Formularz umieszczamy w kodzie strony bezpośrednio po znaczniku <section>. Jego kod został zaprezentowany na listingu 1.7.

Listing 1.7. Kod formularza filtrowania

```
<form action="users.php" method="GET">
 <div class="form-row align-items-center">
  <div class="col-auto">
   <label class="sr-only" for="surname">surname</label>
   <input type="text" class="form-control mb-2"
    id="surname" name="surname" placeholder="Surname">
  </div>
  <div class="col-auto">
   <button type="submit" name="filter" class="btn
    btn-primary mb-2">Filtruj</button>
  </div>
 </div>
</form>
```

Dane z formularza są przesyłane do strony users.php. Należy więc odpowiednio obsłużyć żądanie, tak by filtrowanie zaczęło działać. W tym celu pobierzemy łańcuch znaków wpisany w polu formularza i wykorzystamy go w sekcji *where*

zapytania SQL. Kod tworzący zapytanie zostanie rozbudowany o instrukcję warunkową, tak jak to pokazano na listingu 1.8.

Listing 1.8. Kod tworzący zapytanie SQL z warunkiem WHERE

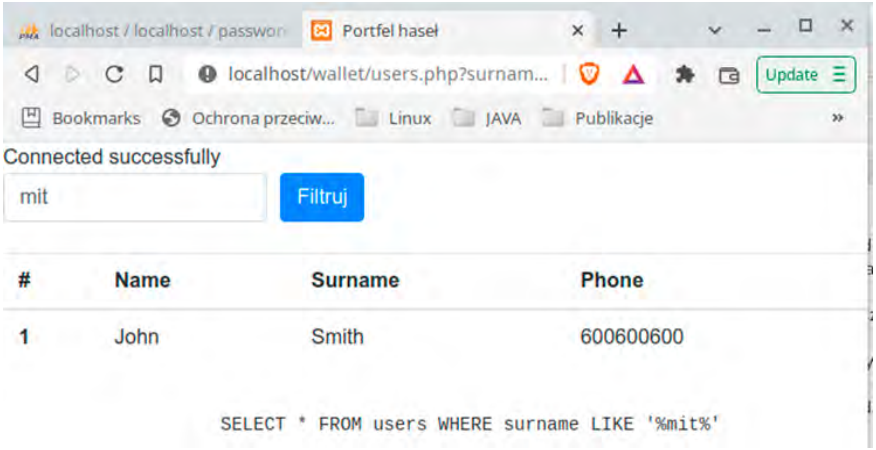
```
$SQL = "SELECT * FROM users";
if(isset($_REQUEST["filter"])){
 $surname=$_REQUEST["surname"];
 $sql= $sql . " WHERE surname LIKE '%". $surname ."%'";
}
```

Do naszej aplikacji dodamy również wyświetlanie treści zapytania bazodanowego, jakie jest wysyłane do bazy. Pomoże to na późniejszych etapach zrozumieć mechanizm wstrzykiwania kodu SQL. Do wyświetlania treści zapytania użyjemy kodu z listingu 1.9.

Listing 1.9. Wyświetlanie treści zapytania SQL

```
<pre><code>
 <?php
  echo $sql
 ?>
</code></pre>
```

Finalnie utworzona strona powinna wyglądać tak, jak zostało to przedstawione na rysunku 1.12.



Rys.1.12. Wygląd strony users.php

1.4. Wprowadzanie danych do bazy

Aplikację z poprzedniego rozdziału rozbudujemy o możliwość dodawania użytkowników. W tym celu utworzymy dodatkową stronę user_add.php, na której umieścimy formularz do wprowadzania danych nowego użytkownika. Dane z formularza będą przekazywane do strony users.php, gdzie dodany zostanie kod obsługi żądania. Jego zadaniem będzie pobranie danych z formularza i zapisanie ich do bazy danych.

Stronę user_add.php oraz zawarty na niej formularz tworzymy analogicznie do formularza wyszukiwania, który umieściliśmy na stronie users.php. Kod strony user_add.php został zaprezentowany na listingu 1.10. Kod obsługujący żądanie dodania nowego użytkownika, który należy dodać na stronie users.php, został przedstawiony na listingu 1.11.

Listing 1.10. Kod strony user_add.php

```
<<!DOCTYPE HTML>
<html lang=”pl”>

<head>
 <meta http-equiv=”Content-Type” content=”text/html;
  charset=utf-8” />
 <meta http-equiv=”Content-Language” content=”pl” />
 <meta name=”description” content=”” />
 <meta name=”keywords” content=”” />
 <link rel=”stylesheet” href=”css/bootstrap.min.css”>
 <title>Portfel haseł</title>
</head>

<body>
 <header></header>
 <section>
  <!--formularz dodania nowego użytkownika-->
  <form action=”users.php” method=”GET”>
   <div class=”form-row align-items-center”>
    <div class=”col-auto”>
     <label class=”sr-only” for=”surname”>surname</label>
     <input type=”text” class=”form-control mb-2”
      id=”surname” name=”surname” placeholder=”Surname”>
    </div>
    <div class=”col-auto”>
     <label class=”sr-only” for=”surname”>name</label>
     <input type=”text” class=”form-control mb-2” id=”name”
```

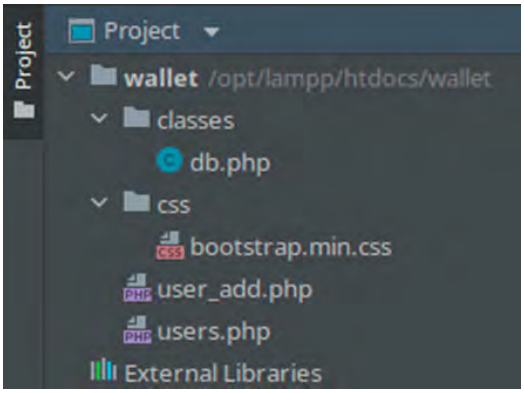
```
      name="name" placeholder="Name">
    </div>
  <div class="col-auto">
   <label class="sr-only" for="login">name</label>
   <input type="text" class="form-control mb-2" id="login"
    name="login" placeholder="login">
  </div>
  <div class="col-auto">
   <label class="sr-only" for="password">password</label>
   <input type="password" class="form-control mb-2"
    id="password" name="password" placeholder="password">
  </div>
  <div class="col-auto">
   <label class="sr-only" for="surname">phone</label>
   <input type="text" class="form-control mb-2" id="phone"
    name="phone" placeholder="phone">
  </div>
  <div class="col-auto">
   <button type="submit" name="user_add" class="btn
    btn-primary mb-2">Dodaj</button>
   </div>
  </div>
 </form>
</section>

<footer></footer>
</body>
</html>
```

Listing 1.11. Kod dodający dane użytkownika do bazy

```
//dodanie nowego użytkownika
if(isset($_REQUEST[“user_add”])){
 $surname="'".$_REQUEST[“surname”]."'";
 $name="'".$_REQUEST[“name”]."'";
 $phone="'".$_REQUEST[“phone”]."'";
 $login="'".$_REQUEST[“login”]."'";
 $password="'".$_REQUEST[“password”]."'";
 $SQL= “insert into users
  (name, surname, phone,login,password)
  values
  ($name,$surname,$phone,$login,$password) “;
 echo $SQL;
 $conn->query($SQL);
```

1.5. Porządkowanie kodu

Do tej pory skupialiśmy się na zaprogramowaniu podstawowych funkcjonalności bez zwracania uwagi na aspekty porządkowania kodu. W efekcie, kod PHP realizujący funkcje dostępu do bazy danych czy też logikę biznesową jest wymieszany z kodem HTML. Znacząco utrudnia to wprowadzanie zmian w kodzie. Dlatego też zajmiemy się podzieleniem kodu na dwie części. Jedna będzie odpowiadała za generowanie widoku, druga za logikę biznesową oraz dostęp do danych. Widok pozostawimy w dotychczasowych plikach. Obsługę dostępu do danych przeniesiemy do oddzielnego pliku. Plik ten umieścimy w nowo utworzonym katalogu *classes*. Katalog ten będzie nam służył do przechowywania plików PHP realizujących logikę biznesową aplikacji oraz dostęp do danych i dodatkowe funkcje. Struktura projektu przyjmie postać przedstawioną na rysunku 1.12.



Rys.1.13. Struktura projektu

Tworzymy plik db.php. Do niego przeniesiemy kod obsługujący połączenie z bazą danych. Kod ten umieścimy wewnątrz klasy o nazwie *DB*. Nawiązanie połączenia z bazą danych zrealizujemy w konstruktorze klasy. Dzięki temu utworzenie obiektu tej klasy będzie realizowało funkcję nawiązania połączenia z bazą. Jako, że nasza aplikacja będzie współpracowała z jedną bazą danych to dane niezbędne do połączenia z bazą (adres serwera, login, hasło, nazwa bazy) umieścimy na stałe w konstruktorze. Za wykonanie operacji na bazie danych będą odpowiadały niezależne metody. Jedną z nich będzie metoda `select($SQL)`, która będzie wykonywała zapytanie przekazane jako parametr i zwracała uzyskane wyniki. Drugą metodą będzie metoda `user_add()`, która jako parametry przyjmie dane użytkownika, po czym skonstruuje zapytanie dodające rekord nowego użytkownika do bazy i wykona je. Kod klasy *DB* został przedstawiony na listingu 1.13.

Listing 1.12. Klasa DB

```
<?php
class DB
{
 private $conn;

 //konstruktor nawiązujący połączenie
 public function __construct()
 {
  $servername = "localhost";
  $username = "root";
  $password = "";
  $dbname = "password_wallet";
  $this->conn = new mysqli($servername, $username,
        $password, $dbname);
  // Check connection
  if ($this->conn->connect_error) {
   die("Connection failed: " . $this->conn->connect_error);
  }
  echo „Connected successfully <BR/>";
 }

 //destruktor zamykający połączenie z bazą
 function __destruct()
 {
  $this->conn->close();
 }

 //wykonanie zapytania przekazanego jako parametr
 public function select($SQL)
 {
  $result = $this->conn->query($SQL);
  return $result;
 }

 //dodanie nowego użytkownika
 public function user_add($surname, $name, $phone, $login,
          $password){
  $surname="'".$surname."'";
  $name="'".$name."'";
  $phone="'".$phone."'";
  $login="'".$login."'";
  $password="'".$password."'";
  $SQL= "insert into users (name, surname,
```

```
        phone,login,password) values
        ($name,$surname,$phone,$login,$password) ";
         echo $SQL;
         $this->conn->query($SQL);
    }
}
```

Po przeniesieniu kodu obsługi bazy danych do klasy *DB* możemy zacząć z niego korzystać na stronie users.php. Dołączamy plik db.php za pomocą dyrektywy include_once do strony users.php, a następnie wywołujemy konstruktor klasy *DB* tak, jak to pokazano na listingu 1.13.

Listing 1.13. Dołączenie pliku db.php i wywołanie konstruktora

```
include_once "classes/db.php";
$db = new DB();
```

Połączenie z bazą danych zostało nawiązane. Możemy przystąpić do modyfikowania kodu strony users.php. Naszym celem jest usunięcie z niej instrukcji wykonujących operacje na bazie danych. Chcemy, aby kod strony korzystał z danych dostarczanych przez klasę *DB*. Dzięki temu łatwiejsze będzie kontrolowanie dostępu do bazy oraz sposobu połączenia z nią. Cały kod odpowiadający za operacje bazodanowe będzie się znajdował w wydzielonej klasie. W kodzie pliku users.php odwołujemy się do metod `select()` oraz `user_add()` pochodzących z klasy *DB*, które pośredniczą w komunikacji pomiędzy stroną a bazą danych. Po wprowadzeniu modyfikacji kod pliku users.php przyjmie postać zaprezentowaną na listingu 1.14. Zauważmy, że część kodu prezentowanej strony pominięto. Zamiast niego na listingu widnieją trzy kropki. Oznacza to, że kod w miejscu kropek jest identyczny z zaprezentowanym na listingach 1.5 i 1.6. Konwencję tę będziemy stosować w dalszej części książki.

Listing 1.14. Strona users.php korzystająca z klasy DB

```
<?php
include_once "classes/db.php";
$db = new DB();
//dodanie nowego użytkownika
if(isset($_REQUEST["user_add"])){
 $surname=$_REQUEST["surname"];
 $name=$_REQUEST["name"];
 $phone=$_REQUEST["phone"];
 $login=$_REQUEST["login"];
 $password=$_REQUEST["password"];
```

```
 $db->user_add($name,$surname,$phone,$login,$password);
}
?>
<!DOCTYPE HTML>
<html lang=”pl”>
 <head...>
 <body>
  <header></header>
  <section>
   <!--formularz filtrowania po nazwisku-->
   <form action=”users.php” method=”GET”>
    <div class=”form-row align-items-center”>
     <div class=”col-auto”>
      <label class=”sr-only” for=”surname”>surname</label>
      <input type=”text” class=”form-control mb-2”
       id=”surname” name=”surname” placeholder=”Surname”>
     </div>
     <div class=”col-auto”>
      <button type=”submit” name=”filter” class=”btn
       btn-primary mb-2”>Filtruj</button>
     </div>
    </div>
   </form>
<?php
$sql = “SELECT * FROM users”;
if(isset($_REQUEST[“filter”])){
 $surname=$_REQUEST[“surname”];
 $sql= $sql . “ WHERE surname LIKE ‘%”. $surname .”%’”;
}
echo $sql .’<BR/>’;
$result = $db->select($sql);
if ($result->num_rows > 0) {
?>
 <table class=”table”>
  <thead...>
  <tbody>
 <?php
 // wyświetlenie danych z wierszy
 while($row = $result->fetch_assoc()) {
 ?>
  <tr...>
 <?php
 }
}
 ?>
```

```
    </tbody>
   </table>
   <pre...>
   <a href=”user_add.php” class=”badge badge-primary”>
    Dodaj użytkownika</a>
  </section>
  <footer></footer>
 </body>
</html>
```

2. Operacje kryptograficzne

Operacje kryptograficzne są niezbędne do ochrony poufności danych. W niniejszym rozdziale omówimy wybrane mechanizmy kryptograficzne, które będą nam potrzebne w dalszej części książki.

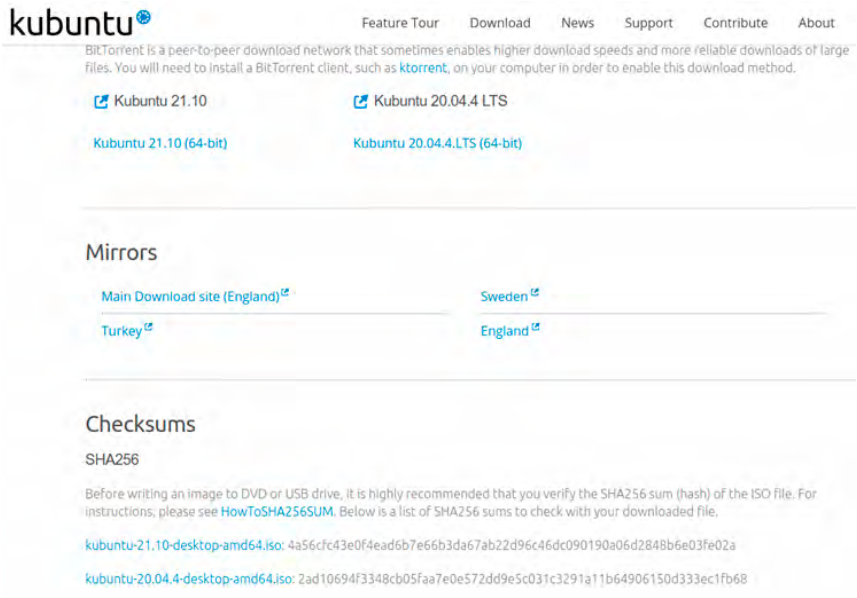
2.1. Skróty kryptograficzne

Obliczanie skrótu kryptograficznego jest operacją polegającą na przekształceniu dowolnego zbioru danych w unikalny ciąg bitów o określonej długości [9]. Funkcje skrótu są jednokierunkowe. Oznacza to, że możliwe jest przekształcenie dowolnego zbioru danych do postaci skrótu kryptograficznego. Nie ma natomiast możliwości wykonania operacji odwrotnej, czyli odtworzenia danych, z których wygenerowany został skrót. Istotny jest również fakt, że wyliczenie skrótu z określonego zbioru danych zawsze da tę samą wartość wynikową. Ze względu na powyższe właściwości, skrótów kryptograficznych używa się do potwierdzania oryginalności danych. Mamy tu na myśli weryfikowanie poprawności kopii zapasowych, gdzie przed odtworzeniem danych z kopii obliczamy skrót kryptograficzny zawartych w niej danych i porównujemy go ze skrótem obliczonym bezpośrednio po wykonaniu tejże kopii. Jeśli skróty są identyczne oznacza to, że dane nie uległy zmianie, czyli są oryginalne. Różniące się skróty wskazują na to, że dane uległy modyfikacji, co może być wynikiem uszkodzenia lub zmodyfikowania danych przechowywanych w kopii.

Skróty kryptograficzne stosowane są również jako narzędzie pozwalające na weryfikowanie pobieranego oprogramowania. Twórcy, oprócz pliku z oprogramowaniem, często udostępniają jego skrót kryptograficzny. Dzięki temu każdy ma możliwość zweryfikowania poprawności pobranego pliku. Wystarczy bowiem samodzielnie obliczyć skrót kryptograficzny pobranego pliku i sprawdzić czy jest identyczny ze skrótem opublikowanym przez twórcę oprogramowania. Przykładem może tu być strona pobierania systemu Kubuntu, na której zamieszczono skróty kryptograficzne udostępnionego oprogramowania, tak jak to pokazano na rysunku 2.1.

Obliczenie wartości skrótu może być realizowane przez dowolny program oferujący tę funkcjonalność. Możemy ją również zaimplementować samodzielnie w kodzie strony. Służy do tego funkcja `hash_file()`. Jej argumentami są: algorytm używany do obliczenia skrótu oraz ścieżka do pliku.

Weryfikacja oryginalności pliku odbywa się poprzez porównanie wcześniej obliczonego skrótu (powinien być trwale zapisany) ze skrótem weryfikowanego pliku. Kod prezentujący sposób wykonania omawianych operacji zaprezentowano na listingu 2.1.



Rys.2.1. Strona pobierania systemu Kubuntu

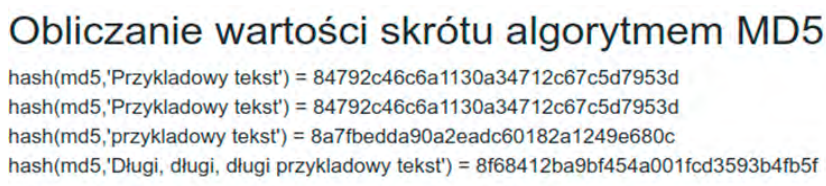
Listing 2.1. Weryfikacja integralności plików

```
<?php
 //obliczenie wartości skrótu pliku oryginalnego
 $hash=hash_file('sha512', 'users.php');

 //sprawdzenie czy plik się nie zmienił
 if (hash_file('sha512', 'users.php')== $hash) {
  echo ,Plik jest oryginalny!<br/>';
 } else {
  echo ,Wykryto modyfikacje w pliku.<br/>';
 }
?>
```

Skróty kryptograficzne są również wykorzystywane do przechowywania haseł użytkowników. Są one powszechnie używane do tego celu ze względu na nieodwracalność operacji obliczenia skrótu. Przechowywanie haseł w postaci skrótów kryptograficznych nie daje możliwości poznania ich postaci jawnej (niezaszyfrowanej) osobom, które są w stanie odczytać je z bazy danych. Mówimy tu zarówno o hakerach jak i administratorach systemu. Zobaczmy jakie wyniki jesteśmy w stanie uzyskać w wyniku obliczenia wartości skrótu wybranych łańcuchów znaków. W tym celu wykonamy kod z listingu 2.2. Wyniki jego działania pokazane zostały na rysunku 2.2.

Listing 2.2. Obliczenie wartości skrótów wybranych łańcuchów znaków

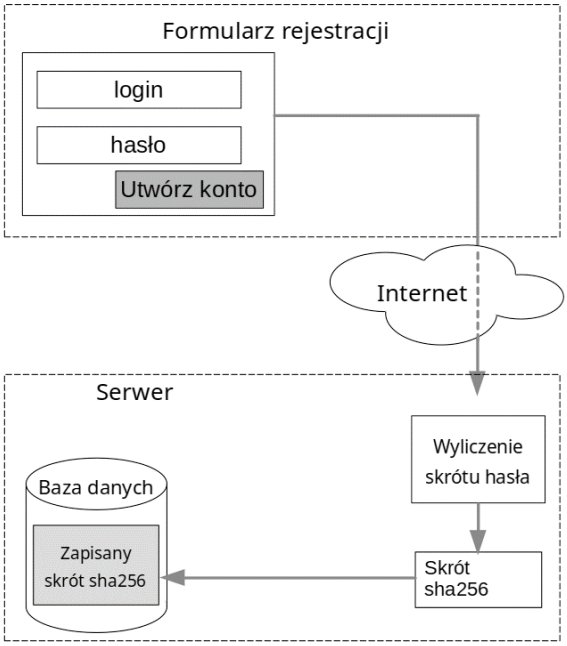
```
<h2>Obliczanie wartości skrótu algorytmem MD5</h2>
<p>
<?php
 $text='Przykladowy tekst';
 $text1='przykladowy tekst';
 $text2='Długi, długi, długi przykladowy tekst';
 echo "hash(md5,'".$text."') = " . hash("md5",$text)
  . "<br/>";
 echo "hash(md5,'".$text."') = " . hash("md5",$text)
  . "<br/>";
 echo "hash(md5,'".$text1."') = " . hash("md5",$text1)
  . "<br/>";
 echo "hash(md5,'".$text2."') = " . hash("md5",$text2)
  . "<br/>";
?>
</p>
```



Rys. 2.2. Obliczone wartości skrótów

Analizując wyniki przedstawione na rysunku 2.2. możemy zauważyć, że długość skrótu kryptograficznego nie zależy od rozmiaru danych wejściowych – jest stała dla każdego z algorytmów. Ponadto widzimy, że wartość skrótu różni się w zależności od danych wejściowych – zmiana nawet jednego bitu powoduje znaczącą zmianę skrótu kryptograficznego. Ze względu na te właściwości skróty kryptograficzne nadają się do przechowywania haseł. Użytkownik, rejestrując się w systemie, podaje swoje dane logowania – login i hasło. Dane te są utrwalane w bazie danych w postaci loginu użytkownika i skrótu kryptograficznego jego hasła, tak jak to pokazano na rysunku 2.3.

Realizacja weryfikacji poprawności hasła wprowadzonego przez użytkownika odbywa się poprzez porównanie wartości skrótu obliczonego z hasła użytkownika ze skrótem przechowywanym w bazie danych. Proces ten przedstawiono na rysunku 2.4.

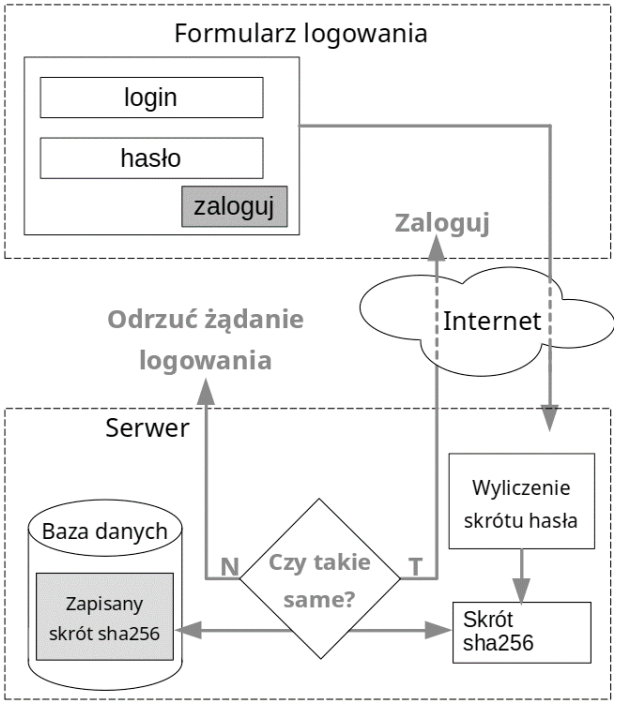


Rys. 2.3. Rejestracja użytkownika

Kod weryfikujący poprawność hasła przedstawiono na listingu 2.3. Aktualnie nie zawiera on funkcjonalności utrwalenia hasła.

Listing 2.3. Weryfikowanie poprawności hasła

```
<h2>
$text='Przykladowy tekst';
$hash=hash("md5",$text);
echo "hash(md5,'".$text."') = " . $hash . "<br/><br/>";
if (hash("md5",$text)== $hash) {
 echo 'Password is valid!<br/>';
}
else {
 echo 'Invalid password.<br/>';
}
</h2>
```



Rys. 2.4. Logowanie użytkownika

2.2. Sól i pieprz w skrótach kryptograficznych

Funkcja skrótu jest funkcją jednokierunkową. Nie da się więc wyliczyć hasła na podstawie skrótu. Jednak da się taki skrót złamać. Możemy zastosować do tego metodę siłową (ang. brute force) polegającą na generowaniu kolejnych haseł, obliczaniu ich skrótów i porównywaniu ich ze skrótem poszukiwanego hasła. Aby zobrazować operację, na listingu 2.4 przedstawiono fragment kodu prezentujący łamanie metodą siłową pinu o długości do 8 cyfr. W pierwszej części kodu generowany jest losowy pin i obliczany jest jego skrót kryptograficzny. Następnie na podstawie skrótu odgadywany jest wylosowany pin. Wynik uruchomienia prezentowanego skryptu na komputerze z procesorem i7-4702MQ, posiadającym taktowanie bazowe 2,2 GHz i taktowanie turbo 3,2GHz, przedstawiono na rysunku 2.5.

Listing 2.4. Łamanie skrótu kryptograficznego metodą siłową

```
<?php
//generowanie kodu do odgadnięcia
$pin_code_to_find=rand(0,99999999);
echo „Pin do odgadnięcia: „.$pin_code_to_find.”<br/>”;
$hash_to_find=hash(“md5”,$pin_code_to_find);

//odgadywanie kodu
//pobranie czasu rozpoczęcia operacji odgadywania
$start = microtime(true);
//wartość początkowa pinu
$pin_code=-1;
$hash=”.”;
//weryfikowanie kolejnych pinów
do {
 $hash=hash(“md5”,$pin_code);
 if($hash==$hash_to_find){
  echo „Poszukiwany pin: „.$pin_code.”<br/>”;
  break;
 }
 $pin_code++;
} while (true);
//obliczenie czasu trwania operacji
$time_elapsed_secs = microtime(true) - $start;
echo “czas operacji [s]: “.$time_elapsed_secs;
?>
```

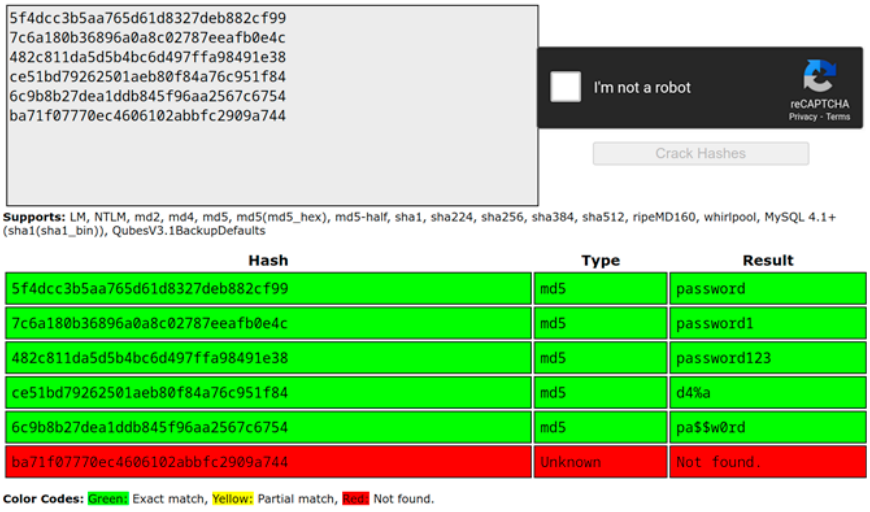
Rys. 2.5. Wynik działania kodu z listingu 2.4

Analiza wyników z rysunku 2.5 pokazuje, że zweryfikowanie znaczącej liczby skrótów kryptograficznych zabiera niewiele czasu. Oczywiście nie możemy pominąć faktu, że korzystamy z algorytmu MD5, który pozwala na szybkie obliczanie skrótów. W przypadku innych algorytmów czas ten może być znacząco dłuższy. Niemniej jednak istnieją inne narzędzia łamania skrótów kryptograficznych. Jednym z nich są tęczowe tablice [6]. Stanowią one bardzo efektywne narzędzie pozwalające na łamanie popularnych haseł. Porównanie wydajności wybranych tęczowych tablic znajdziemy w [10].

Tęczowe tablice możemy zaimplementować samodzielnie, korzystając ze źródeł dostępnych w Internecie lub skorzystać z rozwiązań dostępnych online. W tym opracowaniu użyjemy tablic dostępnych na stronie https://crackstation.net/. Najpierw wygenerujemy skróty kryptograficzne następujących haseł:

- password
- password1
- password123
- d4%a
- pa$$w0rd
- d4%s3SA4

Następnie uzyskane skróty wklejamy do formularza dostępnego na stronie https://crackstation.net/. Odpowiedź uzyskaną ze strony przedstawiono na rysunku 2.6.



Rys. 2.6. Efekt użycia tęczowych tablic

Analizując wyniki zaprezentowane na rysunku 2.6 widzimy, że tęczowe tablice są w stanie bez problemu złamać skróty kryptograficzne często używanych haseł. Nawet, jeśli wprowadzimy do nich popularne modyfikacje. Użyta tablica nie była w stanie złamać skrótu dla losowego hasła. Dla pozostałych haseł odpowiedź została zwrócona natychmiast.

Ze względu na wysoką efektywność narzędzi do łamania skrótów kryptograficznych niezbędne jest stosowanie dodatkowych zabezpieczeń. Jedną z możliwości jest stosowanie soli. Sól jest losowym ciągiem bajtów dołączanym do hasła przed obliczeniem wartości skrótu. Dołączenie jej znacząco utrudnia łamanie haseł, zwłaszcza, że do obliczenia skrótu każdego z haseł generujemy losową wartość

soli. Kod realizujący obliczanie skrótu kryptograficznego hasła z solą pokazano na listingu 2.5. Użyto trzech różnych wartości soli. Dla każdej z nich uzyskana wartość skrótu jest inna pomimo użycia identycznych haseł.

Listing 2.5. Obliczanie wartości skrótu z użyciem soli

```
<?php
$text='haslo';
$salts=array(„qROuJeQoGiL2”,”036qROuJQoL2”,”6qROuJQoGiL2”);
foreach ($salts as $salt){
 echo “hash( sha512,”.$salt.” +”.$text.”') =
   “ . hash(“sha512”,$salt. $text) . „<br/>”;
}
?>
```

Użycie soli znacznie utrudnia łamanie skrótu kryptograficznego. Atakujący nie może użyć dostępnych tablic tęczowych – nie uwzględniają one dołączania soli do haseł. Atak musi więc zostać wykonany metodą siłową – do każdego wygenerowanego hasła dołączamy właściwą mu wartość soli i dopiero wówczas możemy obliczyć skrót kryptograficzny do porównania z łamanym skrótem.

Wartość soli najczęściej jest znana atakującemu. Jest ona bowiem przechowywana razem ze skrótem hasła. Aby jeszcze bardziej utrudnić atak na skróty haseł lub nawet uczynić go niemożliwym stosowany jest pieprz. Jest to losowy ciąg bitów dołączany do hasła z solą. Istotne jest jednak miejsce jego przechowywania. Pieprz jest przechowywany w innym miejscu niż skrót i sól. Dzięki temu atakujący, któremu uda się odczytać skróty oraz wartości soli, nie będzie miał dostępu do wartości pieprzu. Przez to nie będzie w stanie przeprowadzić skutecznego ataku na skróty kryptograficzne.

2.3. Algorytmy obliczania skrótu kryptograficznego

Do obliczenia skrótu kryptograficznego możemy wykorzystać różne funkcje skrótu. Każda z nich charakteryzuje się określoną szybkością pracy oraz zużyciem określonych zasobów (mocy procesora i pamięci). Funkcję należy dobrać do zastosowań. W przypadku obliczania sum kontrolnych kopii zapasowej pożądane będzie osiągnięcie dużej szybkości. W przypadku obliczania skrótów haseł – wprost przeciwnie. Tu chcemy zabezpieczyć się przed atakiem siłowym poprzez zmniejszenie jego efektywności. Do algorytmów polecanych do przechowywania haseł należą [11]:

- Argon2id,

- bcrypt,
- scrypt.

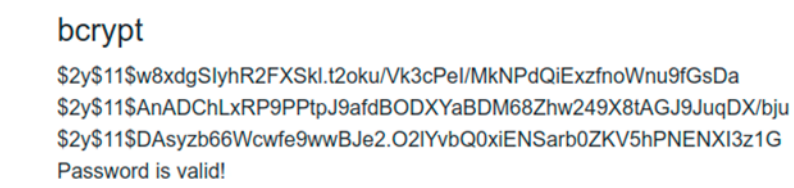
Oprócz nich istnieją inne algorytmy, choć nie zapewniają one tak dobrego poziomu bezpieczeństwa:

- sha3,
- sha2,
- sha1,
- md5,
- RIPE-MD.

Na listingu 2.6 przedstawiono przykład użycia funkcji bcrypt do obliczenia skrótu kryptograficznego. Na rysunku 2.7 zaprezentowano wynik działania kodu zawartego z listingu 2.6. Warto zwrócić uwagę na fakt, że kilkukrotne obliczenie skrótu z tego samego hasła za każdym razem daje inny wynik. Spowodowane jest to tym, że algorytm za każdym razem używa innej, losowej wartości soli. Użyta sól dołączana jest do wygenerowanego skrótu. Do weryfikacji skrótu używana jest funkcja `password_verify()`, która odczytuje użytą wartość soli i wykorzystuje ją do obliczenia skrótu weryfikowanego hasła.

Listing 2.6. Obliczanie wartości skrótu algorytmem bcrypt

```
$text='haslo';
$options = [
 'cost' => 11
];
echo password_hash($text, PASSWORD_BCRYPT,
$options)."<br/>";
echo password_hash($text, PASSWORD_BCRYPT,
$options)."<br/>";
$hash=password_hash($text, PASSWORD_BCRYPT, $options);
echo $hash."<br/>";
if (password_verify($text, $hash)) {
 echo 'Password is valid!';
} else {
 echo 'Invalid password.';
}
```



Rys. 2.7. Skróty kryptograficzne obliczone za pomocą funkcji bcrypt

Temat algorytmów funkcji skrótu podsumujemy za pomocą prostego porównania. Używając kodu z listingu 2.7 porównane zostały czasy łamania kodu pin metodą siłową. Łamanie obejmowało wykonanie 100 operacji obliczenia skrótu kryptograficznego na procesorze i7-4702MQ. Uzyskane wyniki przedstawione zostały na rysunku 2.8. Widzimy na nich, że czas potrzebny do złamania skrótu obliczonego za pomocą algorytmu bcrypt jest znacząco dłuższy niż czas łamania skrótów obliczonych innymi porównywanymi algorytmami. Należy zwrócić uwagę na fakt wykonywania jedynie 100 operacji obliczenia skrótu. W przypadku łamania skrótów mocnych haseł liczba operacji obliczenia skrótu oraz czas całej operacji będą znacząco większe.

Listing 2.7. Kod łamiący skrót kryptograficzny

```
<h4>Łamanie skrótu wyliczonego różnymi algorytmami</h4>
<p>
<?php
//generowanie kodu do odgadnięcia
$pin_code_to_find=100;
echo „Pin do odgadnięcia: „.$pin_code_to_find.”<br/>”;
$algorithms=array(“md5”,”sha512”,”sha256”,”ripemd320”);
foreach ($algorithms as $algo){
 //obliczanie skrótu poszukiwanego hasła
 $hash_to_find=hash($algo,$pin_code_to_find);
 //odgadywanie kodu
 //pobranie czasu rozpoczęcia operacji odgadywania
 $start = microtime(true);
 //wartość początkowa pinu
 $pin_code=0;
 $hash=”.”;
 //weryfikowanie kolejnych pinów
 do {
  $hash=hash($algo,$pin_code);
  if($hash==$hash_to_find){
   break;
  }
  $pin_code++;
  } while (true);
 //obliczenie czasu trwania operacji
 $time_elapsed_secs = microtime(true) - $start;
 echo „czas operacji dla algorytmu „.$algo.” [s]: “.$time_
elapsed_secs. “<br/>”;
}
//dla algorytmu bcrypt
$options = [
 ‘cost’ => 11
```

```
];
//obliczanie skrótu poszukiwanego hasła
$hash_to_find=password_hash($pin_code_to_find, PASSWORD_
BCRYPT, $options);
//odgadywanie kodu
//pobranie czasu rozpoczęcia operacji odgadywania
$start = microtime(true);
//wartość początkowa pinu
$pin_code=0;
//weryfikowanie kolejnych pinów
do {
 if (password_verify($pin_code, $hash_to_find)) {
  break;
 }
 $pin_code++;
} while (true);
//obliczenie czasu trwania operacji
$time_elapsed_secs = microtime(true) - $start;
echo „czas operacji dla algorytmu bcrypt [s]: „.$time_
elapsed_secs. „<br/>”;
?>
</p>
```

Rys. 2.8. Czasy łamania skrótu kryptograficznego

2.4. HMAC

HMAC (ang. Keyed-Hash Message Authentication Code) w uproszczeniu możemy określić jako obliczanie skrótu kryptograficznego z użyciem klucza. Jego zadaniem jest potwierdzenie oryginalności i autentyczności danych. Możliwe jest to dzięki użyciu klucza kryptograficznego podczas operacji obliczenia wartości skrótu HMAC. Dzięki temu tylko posiadacz klucza może zweryfikować

poprawność danych na podstawie otrzymanego skrótu HMAC. Jednocześnie, atakujący nie ma możliwości wygenerowania poprawnego skrótu HMAC nawet jeśli uda mu się podmienić zbiór danych. Do tego niezbędny byłby użyty klucz kryptograficzny. Funkcji HMAC używamy również do ochrony przechowywanych haseł. Jest to funkcja jednokierunkowa, o wiele mniej podatna na wystąpienie kolizji od funkcji skrótu [13]. Przykładowy kod obliczający wartość skrótu HMAC oraz prezentujący sposób porównania skrótów HMAC został przedstawiony został na listingu 2.8.

Listing 2.8. Obliczanie wartości skrótu HMAC

```
$hmac=hash_hmac('sha512', 'Message to be hashed with HMAC',
'secret');
$hmac2=hash_hmac('sha512', 'Message to be hashed with
HMAC', 'secret1');
//weryfikacja czy skróty są takie same
if (hash_equals($hmac, $hmac2) ) {
 echo „skróty identyczne!<br/>";
}
else
 echo „skróty różnia się!<br/>";
```

2.5. Szyfrowanie

Poufność danych chronimy często poprzez ich zaszyfrowanie. Pod pojęciem szyfrowania rozumiemy przekształcenie danych do postaci niezrozumiałej dla osób postronnych. Do szyfrowania używamy wybranego algorytmu kryptograficznego [5, 2]. Listę nazw algorytmów kryptograficznych dostępnych w języku PHP możemy wyświetlić za pomocą kodu widocznego na listingu 2.9.

Listing 2.9. Wyświetlenie listy algorytmów kryptograficznych

```
$methods= openssl_get_cipher_methods();
foreach ($methods as $method) {
 echo $method. „<br/>";
}
```

Przykładowa nazwa algorytmu ma postać: aes-128-cbc. Składa się ona zazwyczaj z trzech części rozdzielonych myślnikami. Pierwsza część określa zastosowany algorytm szyfrowania. Druga mówi o używanym rozmiarze bloku. Trzecia definiuje tryb pracy algorytmu kryptograficznego.

W operacji szyfrowania używamy klucza kryptograficznego – tajnej sekwencji bitów, bez której nie uda się wywołać operacji deszyfrowania. Jeśli do funkcji deszyfrującej podamy jako parametr niewłaściwy klucz, operacja deszyfrowania nie powiedzie się, a funkcja deszyfrująca nie zwróci wyniku. Oczywiście możliwe jest łamanie klucza kryptograficznego, choćby metodą siłową polegającą na próbie deszyfrowania każdym możliwym kluczem. Jednak czasochłonność takich metod jest bardzo duża. Przykład prostego szyfrowania algorytmem 3DES pracującym w trybie ECB (elektronicznej książki kodowej) przedstawiono na listingu 2.10. Niestety, takie szyfrowanie nie zapewnia pożądanego poziomu bezpieczeństwa ze względu na wybrany tryb pracy algorytmu. W trybie tym szyfrowane dane dzielone są na bloki, a każdy blok szyfrowany niezależnie. W przedstawionym przykładzie dane zawierają powtarzający się ciąg znaków. Analizując dane zaszyfrowane zauważymy również występowanie powtórzeń – odpowiadają one powtórzeniom w tekście jawnym.

Listing 2.10. Szyfrowanie tekstu algorytmem 3DES w trybie ECB

```
//tekst do zaszyfrowania
$plaintext = "message message message message message mes-
sage message message message message ";
//algorytm
$cipher = „des-ede";
//sprawdzenie czy podany algorytm jest obsługiwany przez
openssl
if (in_array($cipher, openssl_get_cipher_methods()))
{
 //wygenerowanie klucza kryptograficznego
 $key = openssl_random_pseudo_bytes(16);
 //szyfrowanie
 $ciphertext = openssl_encrypt($plaintext, $cipher, $key);
 //deszyfrowanie
 $original_plaintext = openssl_decrypt($ciphertext,
$cipher, $key);
 echo $ciphertext."<br/>";
 echo $original_plaintext."<br/>";
}else
 echo „podaj poprawną nazwę algorytmu szyfrującego<br/>";
```

Podniesienie poziomu bezpieczeństwa oferowanego przez operację szyfrowania wymaga zastosowania odpowiedniego trybu pracy algorytmu kryptograficznego. Na listingu 2.11 przedstawiono przykład szyfrowania z użyciem algorytmu aes-128-gcm. Oferuje on wysoki poziom bezpieczeństwa. Powtórzenia w szyfrogramie nie wystąpią tu, ze względu na sumowanie każdego bloku jawnego z wartością licznika. Licznikiem nazywamy tu ciąg binarny o rozmiarze bloku algorytmu

kryptograficznego, który jest zwiększany po zaszyfrowaniu każdego bloku. Tryb GCM dodatkowo generuje znacznik autentyczności (ang. authentication tag), który pozwala na zweryfikowanie autentyczności zaszyfrowanych danych [4].

Listing 2.11. Szyfrowanie tekstu algorytmem AES pracującym w trybie GCM

```
$plaintext = "message message message message message mes-
sage message message message message message message ";
//algorytm
$cipher = „aes-128-gcm";
//sprawdzenie czy podany algorytm jest obsługiwany przez
openssl
if (in_array($cipher, openssl_get_cipher_methods())) {
 //wygenerowanie klucza kryptograficznego
 $key = openssl_random_pseudo_bytes(16);
 //określenie wymaganej długości wektora inicjalizacyjnego
 $ivlen = openssl_cipher_iv_length($cipher);
 //wygenerowanie wektora inicjalizacyjnego
 $iv = openssl_random_pseudo_bytes($ivlen);
 //szyforwanie
 $ciphertext = openssl_encrypt($plaintext, $cipher, $key,
$options = 0, $iv, $tag);
 //deszyfrowanie
 $original_plaintext = openssl_decrypt($ciphertext,
         $cipher, $key, $options = 0, $iv, $tag);
 echo $ciphertext . "<br/>";
 echo $original_plaintext . "<br/>";
}else
 echo „podaj poprawną nazwę algorytmu szyfrującego<br/>";
```

Rekomendacje OWASP (ang. Open Web Application Security Project) wskazują, że w celu zapewnienia bezpieczeństwa zaszyfrowanych danych powinniśmy używać algorytmu AES o rozmiarze bloku nie mniejszym niż 128 bitów. Zalecane jest stosowanie bloków o rozmiarze 256 bitów. Rekomendowane tryby pracy to GCM oraz CCM, a w przypadku ich niedostępności CTR lub CBC [12].

2.6. Zadanie do samodzielnego wykonania

Utwórz stronę password_add.php, która będzie dodawała hasła do portfela. Dodane hasła zapisz w tabeli passwords w postaci zaszyfrowanej. Jako klucza szyfrującego użyj skrótu MD5 obliczonego z hasła używanego przez użytkownika do logowania się do aplikacji.

3. Autentykacja użytkownika

Terminem 'autentykacja' określamy proces uwierzytelnienia użytkownika w systemie informatycznym, czyli potwierdzenia jego tożsamości[7]. Autentykacja umożliwia ograniczenie dostępu do systemu tylko dla upoważnionych użytkowników. Niemniej jednak, w przypadku niewłaściwie wykonanej procedury uwierzytelnienia atakujący mogą uzyskać dostęp do systemu. Zagrożenie to jest na tyle poważne, że znalazło się na siódmym miejscu w rankingu 10 najpoważniejszych zagrożeń aplikacji internetowych publikowanym przez organizację OWASP. Ranking dostępny jest pod adresem https://owasp.org/Top10/.

Weryfikacja tożsamości użytkownika możliwa jest na podstawie:

- tego co wie,
- tego czym jest,
- tego co posiada.

Możemy więc opierać się na wiedzy użytkownika. W tym przypadku, najczęściej, użytkownik proszony jest o podanie hasła. Jest to najprostsze zabezpieczenie i jednocześnie najsłabsze ze względu na łatwość podsłuchania hasła czy też jego przechwycenia. Druga z możliwości wykorzystuje cechy użytkownika – najczęściej biometrię. Skanowane są odciski palców, źrenica, rozpoznawana jest twarz lub głos. Możliwe jest też wykorzystanie posiadanych przez użytkownika zasobów. Mogą być to zasoby fizyczne, takie jak karty inteligentne, identyfikatory, tokeny, klucze sprzętowe. Mogą również być wykorzystywane rozwiązania programowe – usługi uwierzytelniania (np. Google Authenticator) czy dodatkowe aplikacje.

Każde z wymienionych rozwiązań może posłużyć do uwierzytelnienia użytkownika. Jeśli wykorzystujemy tylko jedną z nich, mówimy wówczas o uwierzytelnianiu jednoskładnikowym. Obecnie, ze względu na ryzyko przechwycenia lub sfałszowania składnika logowania, coraz częściej stosujemy uwierzytelnianie wieloskładnikowe. Wymaga ono zweryfikowania użytkownika za pomocą więcej niż jednego składnika uwierzytelnienia. Przykładem może tu być uwierzytelnianie dwuskładnikowe stosowane w bankowości elektronicznej. Aby użytkownik mógł się zalogować najpierw musi podać swój login i hasło. Jeśli pomyślnie przejdzie przez ten etap, proszony jest o wykonanie dodatkowej operacji – może to być potwierdzenie chęci logowania w aplikacji mobilnej lub podanie kodu przesłanego na numer telefonu skojarzony z kontem użytkownika.

3.1. Logowanie jednoskładnikowe

Logowanie jednoskładnikowe najczęściej opiera się na zweryfikowaniu loginu i hasła użytkownika. Jeśli są one zgodne z danymi przechowywanymi w bazie

danych użytkownik jest uwierzytelniany. Jest to prosty mechanizm. Jednak wymaga on właściwego zabezpieczenia haseł przechowywanych w systemie. Jest ono konieczne ze względu na ryzyko związane z tym, że atakujący może być w stanie włamać się do bazy danych i odczytać zgromadzone tam dane. Ponadto, hasła użytkowników powinny być przechowywane w sposób uniemożliwiający ich odczyt administratorom systemu. Najczęściej używa się następujących form przechowywania haseł użytkowników:

- skrót kryptograficzny,
- HMAC,
- tekst zaszyfrowany.

Rekomendowaną przez OWASP formą przechowywania haseł użytkowników jest skrót kryptograficzny obliczony algorytmem Argon2id lub bcrypt. W tej książce wybrany został bcrypt ze względu na dostępność algorytmu również w starszych wersjach PHP. Będziemy potrzebowali funkcji obliczających i weryfikujących skrót kryptograficzny. Umieścimy je w osobnej klasie o nazwie *Crypto*. Pozwoli to na zachowanie przejrzystości kodu oraz ułatwi wprowadzanie modyfikacji. W klasie *Crypto* umieścimy dwie funkcje:

- `calculate_bcrypt` – obliczającą skrót kryptograficzny hasła przekazanego, jako parametr wywołania funkcji,
- `verify_bcrypt` – weryfikującą poprawność hasła użytkownika i zwracającą wartość logiczną informującą czy hasło jest zgodne ze skrótem przekazanym do funkcji.

Kod klasy *Crypto* został przedstawiony na listingu 3.1.

Listing 3.1. Klasa Crypto

```
<?php
class Crypto
{
 public function __construct(){}

 function calculate_bcrypt($password)
 {
  $options = [
   'cost' => 12
  ];
  return password_hash($password, PASSWORD_BCRYPT,
           $options);
 }
 function verify_bcrypt($password,$hash){
  if (password_verify($password, $hash)) {
   return true;
```

```
  } else {
   return false;
  }
 }
}
```

Kolejnym krokiem jest zmodyfikowanie funkcjonalności dodawania nowego konta użytkownika tak, aby hasła były zapisywane w postaci skrótu kryptograficznego.

Dodawanie konta użytkownika zostało zrealizowane w funkcji `user_add()` klasy DB. Zmodyfikujemy więc kod klasy tak, aby hasła przed utrwaleniem były przekształcane do postaci skrótu kryptograficznego. Dołączamy za pomocą instrukcji `include_once` plik crypto.php do klasy *DB*. Tworzymy w niej zmienną, która posłuży do przechowywania obiektu klasy Crypto, a w konstruktorze inicjalizujemy zmienną. Modyfikacje wprowadzone do klasy *DB* zaprezentowano na listingu 3.2.

Listing 3.2. Modyfikacja klasy DB

```
<?php
//dołączenie pliku crypto.php
include_once „crypto.php”;

class DB
{
 private $conn;
 //zmienna do przechowywania obiektu klasy Crypto
 private $crypto;

 public function __construct()
 {
  ...
  //inicjalizacja obiektu klasy Crypto
  $this->crypto=new Crypto();
 }
...
```

W klasie *DB* możemy już korzystać z funkcji dostarczanych przez klasę *Crypto*. Modyfikujemy więc metodę `user_add()` tak, aby hasło użytkownika zostało przekształcone w skrót kryptograficzny przed jego zapisaniem w bazie danych, jak to przedstawiono na listingu 3.3. Ponadto, do metody dodamy sprawdzenie czy użytkownik nie podał loginu, który jest już używany. W celu jednoznacznej identyfikacji użytkownika loginy muszą być niepowtarzalne.

Listing 3.3. Modyfikacja metody user_add()

```
public function user_add($surname, $name, $phone, $login,
          $password){
 $surname="'".$surname."'";
 $name="'".$name."'";
 $phone="'".$phone."'";
 $login="'".$login."'";
 $password="'".$this->crypto->calculate_bcrypt(
            $password)."'";
 $sql="SELECT id from users where login=".$login;
 $result = $this->conn->query($sql);
 //sprawdzenie czy login użytkownika nie został już użyty
 if($result->num_rows==0) {
  $sql = "insert into users
      (name, surname, phone, login, password)
      values ($name $surname, $phone, $login, $password)";
  echo $sql;
  $this->conn->query($sql);
  echo „KONTO UTWORZONE";
 }
 else
  echo „BŁĄD TWORZENIA KONTA, PODANY LOGIN JEST ZAJĘTY";
}
```

Możemy utworzyć dodatkowe konta użytkowników. Posłużą nam one do zweryfikowania, czy funkcjonalność logowania działa poprawnie.

Funkcjonalność logowania użytkownika zrealizujemy za pomocą formularza umieszczonego na stronie głównej. Dołączamy do naszego projektu plik dashboard.php, a w nim umieszczamy formularz logowania pozwalający na wpisanie loginu i hasła. Pole hasła powinno być maskowane, czyli wpisywane w nie znaki powinny się wyświetlać w postaci kropek lub innych symboli zastępczych. Maskowanie realizujemy poprzez ustawienie właściwości type elementu input na wartość password. Formularz powinien przekazywać dane do strony dashboard.php.

Dane otrzymane z formularza należy obsłużyć w celu zrealizowania logowania użytkownika. Jeśli logowanie się powiedzie, musimy zapamiętać ten fakt. Posłuży do tego mechanizm sesji pozwalający na przechowanie stanu aplikacji. Na każdej stronie, na której potrzebujemy dostępu do sesji, powinniśmy taką sesję rozpocząć. Dokonujemy tego umieszczając na początku kodu strony instrukcję `session_start()`. Powoduje ona utworzenie nowej sesji lub wznowienie istniejącej sesji na podstawie jej identyfikatora. Identyfikator sesji może być przekazany w treści żądania POST lub GET albo przechowany w ciasteczku (ang. cookie) [14].

Następnie, na stronie dashboard.php dołączamy plik db.php i tworzymy nowy obiekt klasy *DB*. Pozwoli nam to korzystać z funkcjonalności dostępu do bazy danych oferowanej przez tę klasę.

Dane otrzymane z formularza użytkownika odczytujemy poprzez pobranie ich ze zmiennej `_REQUEST` i weryfikujemy ich poprawność. W celu utrzymania przejrzystości kodu funkcjonalność tę umieścimy w metodzie `user_login()` klasy *DB*, tak jak to zaprezentowano na listingu 3.4.

Listing 3.4. Metoda user_login

```
public function user_login($login,$password){
 $login="'".$login."'";
 //pobranie z bazy danych informacji o użytkowniku
 $sql="SELECT id,login,password from users where
       login=".$login;
 $result = $this->conn->query($sql);
 //sprawdzenie czy tylko jeden rekord został zwrócony
 if($result->num_rows==1){
  //pobranie rekordu i zapisanie w tablicy asocjacyjnej
  $row = $result->fetch_assoc();
  //weryfikacja hasła
  if($this->crypto->verify_bcrypt(
        $password,$row[,password'])){
   //ustawienie zmiennych sesji
   $_SESSION['login']=$row['login'];
   $_SESSION['logged_in']=true;
   //wygenerowanie nowego identyfikatora sesji
   session_regenerate_id();
   return true;
  }
 }
 return false;
}
```

Metoda `user_login()` pobiera jako parametry login i hasło podane przez użytkownika. Sprawdza, czy w bazie istnieje rekord użytkownika o podanym loginie. Jeśli tak, sprawdza poprawność podanego hasła poprzez weryfikację skrótu kryptograficznego. Po pomyślnym zweryfikowaniu użytkownika ustawiane są zmienne sesyjne przechowujące dane o zalogowanym użytkowniku. Zestaw danych dobierany jest w zależności od potrzeb. Finalnie, generowany jest nowy identyfikator sesji. Pozwala to zapobiec atakom, w których atakujący przesyła ofierze link do strony logowania zawierający identyfikator sesji (ang. session fixation) [15]. Jeśli ofiara zaloguje się używając otrzymanego identyfikatora sesji, a nie

zostanie on zmieniony, to zarówno ofiara, jak i atakujący zostaną zidentyfikowani jako ten sam zalogowany użytkownik. Dzieje się tak ze względu na to, że numer sesji identyfikujący użytkownika jest taki sam u atakowanego i atakującego.

Przedstawione mechanizmy działają po stronie serwera. Potrzebny będzie jeszcze interfejs użytkownika. Umieścimy go w pliku dashboard.php. Będzie on wyświetlał użytkownikom niezalogowanym formularz logowania, a użytkownikom zalogowanym przycisk wyloguj. Dodatkowo, umieścimy w nim kod odpowiedzialny za obsługę żądań logowania i wylogowania. Przykładowy kod strony dashboard.php przedstawiono na listingu 3.5.

Listing 3.5. Strona dahsboard.php

```
<?php
session_start();
include_once "classes/db.php";
$db = new DB();
//logowanie użytkownika
if(isset($_REQUEST["sign_in"])){
 $login=$_REQUEST["login"];
 $password=$_REQUEST["password"];
 if($db->user_login($login,$password))
  echo „LOGOWANIE POMYŚLNE <BR/>";
 else
  echo „BŁĄD LOGOWANIA <BR/>";
 }
//wylogowanie użytkownika
if(isset($_REQUEST["sign_out"])){
 session_unset();
 session_destroy();
}
?>
<!DOCTYPE HTML>
<html lang="pl">
<head>
 <meta http-equiv="Content-Type" content="text/html;
    charset=utf-8" />
 <meta http-equiv="Content-Language" content="pl" />
 <meta name="description" content="" />
 <meta name="keywords" content="" />
 <link rel="stylesheet" href="css/bootstrap.min.css">
 <title>Portfel haseł</title>
</head>

<body>
 <header></header>
```

```
 <section>
  <!--sprawdzenie czy zalogowany-->
  <?php
  if(isset($_SESSION)&&($_SESSION['logged_in']==true)){
   echo „ZALOGOWANY”;
   //formularz wylogowania użytkownika
   ?>
   <form action=”dashboard.php” method=”POST”>
    <div class=”col-auto”>
     <button type=”submit” name=”sign_out” class=”btn
         btn-primary mb-2”>Wyloguj</button>
    </div>
   </div>
  </form>
  <?php
  }
  else {
   ?>
    <!--formularz logowania użytkownika-->
    <form action=”dashboard.php” method=”POST”>
     <div class=”form-row align-items-center”>
      <div class=”col-auto”>
       <label class=”sr-only” for=”login”>name</label>
       <input type=”text” class=”form-control mb-2”
         id=”login” name=”login” placeholder=”login”>
      </div>
      <div class=”col-auto”>
       <label class=”sr-only” for=”password”>
         password</label>
       <input type=”password” class=”form-control mb-2”
         id=”password” name=”password”
         placeholder=”password”>
      </div>
      <div class=”col-auto”>
       <button type=”submit” name=”sign_in” class=”btn
         btn-primary mb-2”>Zaloguj</button>
      </div>
     </div>
    </form>
   <?php
  }
  ?>
  </section>
  <footer></footer>
 </body>
</html>
```

Przyjrzyjmy się funkcjonalności wylogowania użytkownika. Znajdziemy ją na listingu 3.5, opatrzoną komentarzem. Prawidłowe wylogowanie użytkownika polega na usunięciu wszystkich danych sesji, które są wykorzystywane przez aplikację. Usuwamy więc wpisy w tablicy `_SESSION` oraz finalnie likwidujemy sesję poleceniem `session_destroy()`.

3.2. Logowanie wieloskładnikowe

Logowanie wieloskładnikowe znacząco podnosi poziom bezpieczeństwa. Drugim składnikiem, często wykorzystywanym w tego typu rozwiązaniach, jest kod jednorazowy. Aby jednak spełnił on swoją rolę musi zostać spełniony warunek użycia niezależnych kanałów komunikacyjnych. Oznacza to, że kod musi zostać przekazany innym kanałem niż WWW.

Jednym z ciekawszych rozwiązań jest stworzenie dedykowanej aplikacji, przeznaczonej na urządzenia mobilne, w której użytkownik potwierdza, że to on loguje się na swoje konto poprzez przeglądarkę. Zaletą tego rozwiązania jest całkowite uniezależnienie drugiego etapu autentykacji od aplikacji internetowej. Jeśli jednak nie możemy opracować aplikacji mobilnej dedykowanej logowaniu to możemy użyć kodów jednorazowych. Taki kod generowany jest po pomyślnym przejściu użytkownika przez pierwszy etap autentykacji i dostarczany użytkownikowi niezależnym kanałem: sms, e-mail. O ile wysłanie wiadomości sms wiąże się z koniecznością podłączenia się do bramki sms, o tyle wysłanie wiadomości e-mail jest całkiem prostą czynnością. Skorzystamy z niej prezentując technikę wykonania logowania dwuskładnikowego.

Logowanie, które opracujemy będzie się składało z dwóch etapów:

1) weryfikacji loginu i hasła – będzie to niemal identyczna funkcja do tej, z którą zapoznaliśmy się w poprzednim rozdziale; nie zakończy się ona jednak zalogowaniem użytkownika, lecz przesłaniem użytkownikowi wygenerowanego kodu jednorazowego,
2) weryfikacji kodu jednorazowego – jeśli podany kod będzie poprawny, to użytkownik zostanie zalogowany.

Wykonanie uwierzytelnienia dwuskładnikowego rozpoczniemy od modyfikacji bazy danych. Używając kodu z listingu 3.6 dodamy do tabeli *users* dwa pola:

- *security_code* – służące do przechowywania kodu jednorazowego,
- *security_code_timeout* – używane do przechowywania informacji o czasie ważności kodu.

Listing 3.6. Modyfikacja tabeli users

```
alter table users add security_code integer;
alter table users add security_code_timeout timestamp;
```

Następnie modyfikujemy stronę dashboard.php. Dane z istniejącego formularza logowania będziemy przesyłać do nowo utworzonej strony login_code.php a kod obsługujący logowanie jednoskładnikowe (oznaczony na listingu 3.5 komentarzem *//logowanie użytkownika*) usuniemy. Strona login_code.php będzie przyjmować dane z formularza logowania i wywoływać funkcję weryfikującą podane: login i hasło. Jeśli weryfikacja się powiedzie, strona będzie wyświetlać formularz pozwalający na wprowadzenie kodu jednorazowego. W przeciwnym razie wyświetlony zostanie komunikat błędu. Kod strony login_code.php przedstawiony został na listingu 3.7.

Listing 3.7. Strona login_code.php

```
<?php
session_start();

include_once "classes/db.php";
$db = new DB();
?>
<!DOCTYPE HTML>
<html lang="pl">

 <head>
  <meta http-equiv="Content-Type" content="text/html;
     charset=utf-8" />
  <meta http-equiv="Content-Language" content="pl" />
  <meta name="description" content="" />
  <meta name="keywords" content="" />
  <link rel="stylesheet" href="css/bootstrap.min.css">
  <title>Portfel haseł</title>
 </head>

 <body>
  <header></header>
  <section>
  <?php
  //logowanie użytkownika - pierwszy etap
  if(isset($_REQUEST["sign_in"])){
   $login=$_REQUEST["login"];
   $password=$_REQUEST["password"];
   if($db->user_login_2F_step1($login,$password)){
    //udało się pomyślnie przejść przez pierwszy etap
     uwierzytelnienia
?>
    <!--formularz pobrania kodu od użytkownika-->
```

```
    <p> Podaj kod, który otrzymałeś na adres email
       przypisany do konta, na które się logujesz.</p>
    <form action="dashboard.php" method="POST">
     <div class="form-row align-items-center">
      <div class="col-auto">
       <label class="sr-only" for="code">name</label>
       <input type="text" class="form-control mb-2"
         id="code" name="code" placeholder="code">
       </div>
       <div class="col-auto">
        <button type="submit" name="sign_in" class="btn
         btn-primary mb-2">Wyślij</button>
       </div>
      </div>
     </form>
<?php
   }
   else
    echo „BŁĄD LOGOWANIA <BR/>";
  }
  ?>
  </section>
  <footer></footer>
 </body>
</html>
```

Funkcją weryfikującą poprawność loginu i hasła jest `user_login_2F_step1()`. Jej kod został przedstawiony na listingu 3.8. Funkcja ta weryfikuje poprawność loginu i hasła. Jeśli weryfikacja zakończy się pozytywnie, wówczas generowany jest jednorazowy kod, który jest przesyłany użytkownikowi na adres e-mail skojarzony z kontem. Jednocześnie, kod ten utrwalany jest w bazie danych wraz ze znacznikiem czasu określającym jego ważność. Znacznik ten pozwala na ograniczenie czasu, w którym możliwe jest wykorzystanie kodu. Zapobiegnie to sytuacjom, w których atakujący będzie mógł wykorzystać wcześniej wygenerowany kod do zalogowania się na konto użytkownika. Ponadto, w sesji zapisywany jest bieżący stan procesu logowania wraz z niezbędnymi danymi użytkownika.

Listing 3.8. Funkcja user_login_2F_step1()

```
public function user_login_2F_step1($login,$password){
 $login="'".$login."'";
 //pobranie z bazy danych informacji o użytkowniku
 $sql="SELECT id,login,password,email from users where
       login=".$login;
```

```
 $result = $this->conn->query($sql);
 //sprawdzenie czy tylko jeden rekord został zwrócony
 if($result->num_rows==1){
  //pobranie rekordu i zapisanie w tablicy asocjacyjnej
  $row = $result->fetch_assoc();
  //weryfikacja hasła
if($this->crypto->verify_bcrypt(
    $password,$row[,password'])){
   //pomyślnie zakończono pierwszy etap autentykacji
   $security_code=random_int(100000,999999);
   //wyślij kod e-mailem
   $this->mailer->send_email($row['email'],$security_code);
   //oblicz czas ważności kodu (5minut od momentu wysłania)
   $code_lifetime = date('Y-m-d H:i:s', time() + 300);
   //zapisz kod w bazie
   $sql = „update users set security_code="
     .$security_code.", security_code_timeout='"
     .$code_lifetime."' where id=".$row['id'];
   //echo "<BR/>".$sql."<BR/>";
   $this->conn->query($sql);
   //zapisz w sesji bieżący stan logowania
   $_SESSION['logged_in']='after_first_step';
   $_SESSION['user_id']=$row['id'];
   $_SESSION['login']=$row['login'];
   session_regenerate_id();
   return true;
  }
 }
 return false;
}
```

Wysyłanie wiadomości e-mail wykonane zostało przy użyciu biblioteki *PHPMailer*. Jest ona dostępna wraz z dokumentacją i instrukcjami instalacji i użycia na stronie serwisu github [16]. Bibliotekę należy pobrać, a jej pliki umieścić w katalogu tworzonej aplikacji.

Utworzymy klasę *Mailer*. Będzie on przeznaczona do obsługi procesu wysyłania wiadomości zawierających kody jednorazowe. Wysyłka będzie realizowana za pomocą poczty elektronicznej Do klasy *Mailer* dołączymy wszystkie wymagane zasoby i umieścimy w niej funkcję `send_email()`, która będzie odpowiedzialna za wysłanie wiadomości. Adres e-mail adresata oraz treść wiadomości będą przekazywane do funkcji jako parametr. Wewnątrz funkcji należy podać dane niezbędne do nawiązania połączenia z serwerem poczty elektronicznej oraz utworzyć i wysłać wiadomość. Kod klasy *Mailer* wraz z niezbędnymi komentarzami został zaprezentowany na listingu 3.9.

Listing 3.9. Klasa Mailer

```
<?php
namespace PHPMailer\src\Exception;
use PHPMailer\PHPMailer\Exception;
use PHPMailer\PHPMailer\PHPMailer;
//dołączenie wymaganych plików
require ,./classes/PHPMailer/src/Exception.php';
require ,./classes/PHPMailer/src/PHPMailer.php';
require ,./classes/PHPMailer/src/SMTP.php';

class Mailer
{
 //funkcja wysyłająca wiadomość e-mail
 public  function send_email($address,$content){
  try
  {
   //utworzenie obiektu klasy PHPMailer
   $mail = new PHPMailer;
   //wybranie protokołu SMTP do wysłania wiadomości
   $mail->isSMTP();
   //adres serwera poczty (SMTP)
   $mail->Host = ,poczta.o2.pl';
   //włączenie autentykacji SMTP
   $mail->SMTPAuth = true;
   //nazwa użytkownika
   $mail->Username = ,nadawca@o2.pl';
   //hasło użytkownika
   $mail->Password = ,haslo_do_konta_email';
   //włączenie szyfrowania TLS
   $mail->SMTPSecure = ,tls';
   //wartość w polu „od"
   $mail->From = ,nadawca@o2.pl';
   $mail->FromName = ,OTP source';
   //adresat wiadomości
   $mail->addAddress($address);
   //zawijanie wierszy po 50 znakach
   $mail->WordWrap = 50;
   //wysłanie wiadomości w formacie HTML
   $mail->isHTML(true);
   //temat wiadomości
   $mail->Subject = ,Your security code';
   //treść wiadomości
   $mail->Body = ,This is your authentication code
      <B>'.$content.'</B>';
```

```
   $mail->AltBody = 'This is your authentication code
      ,.$content.'';
   //wysłanie wiadomości
   if (!$mail->send()){
    echo 'Message could not be sent.<BR/>';
    echo ,Mailer Error: , . $mail->ErrorInfo."<BR/>";
   }
   else {
    echo ,Message has been sent <BR/>';
   }
  }catch(Exception $e){
   echo "Exception  " . $e->getMessage()."<BR/>";
   }
 }
}
```

Aby możliwe było użycie funkcji `send_email()` wewnątrz klasy *DB,* należy dołączyć do niej plik zawierający klasę *Mailer* oraz utworzyć zmienną przechowującą obiekt tejże klasy, tak jak to zaprezentowano na listingu 3.10.

Listing 3.10. Dołączenie mailera do klasy DB

```
<?php
...
//dołączenie pliku mailer.php
require_once „mailer.php";

class DB
{
 ...
 //zmienna do przechowywania obiektu klasy Mailer
 private $mailer;

 public function __construct()
 {
 ...
  //inicjalizacja obiektu klasy Mailer
  $this->mailer=new \PHPMailer\src\Exception\Mailer();
 }
```

Przejście pierwszego etapu autentykacji powoduje wyświetlenie formularza przeznaczonego do wprowadzenia kodu jednorazowego, który użytkownik otrzymuje w wiadomości e-mail, tak jak to zaprezentowano na listingu 3.7. Dane z formularza przekazywane są do strony dashboard.php, gdzie kod jest odczytywany.

Fragment kodu PHP odpowiedzialny za wykonanie tej czynności pokazany został na listingu 3.11.

Listing 3.11. Fragment kodu strony dashboard.php

```
//logowanie użytkownika
if(isset($_REQUEST[“sign_in”])){
 $code=$_REQUEST[„code”];
 if($db->user_login_2F_step2($code))
  echo „LOGOWANIE POMYŚLNE <BR/>”;
 else
  echo „BŁĄD LOGOWANIA <BR/>”;
}
```

Kod jednorazowy przekazywany jest do funkcji `user_login_2F_step2()`, która odpowiada za weryfikację jego poprawności. Funkcja ta przyjmuje jako parametr kod podany przez użytkownika. Następnie weryfikuje stan sesji sprawdzając, czy użytkownik pomyślnie zakończył pierwszy etap logowania. Jeśli tak, z bazy danych pobierane są dane użytkownika zapamiętanego w sesji. Jeśli dane użytkownika są prawidłowe a podany kod poprawny, następuje sprawdzenie czy kod nie wygasł. W przypadku spełnienia wszystkich warunków, użytkownik jest logowany do systemu, a informacja o pomyślnym zalogowaniu użytkownika zapisywana jest w zmiennej sesji. Kod funkcji `user_login_2F_step2()` został przedstawiony na listingu 3.12.

Listing 3.12. Funkcja user_login_2F_step2

```
//dwuskładnikowe logowanie użytkownika - etap drugi
public function user_login_2F_step2($code){
 //zweryfikowanie czy użytkownik pomyślnie ukończył
   pierwszy etap logowania
 if($_SESSION[‘logged_in’]==”after_first_step”){
  $code=”’”.$code.”’”;
  //pobranie z bazy danych informacji o użytkowniku i
    weryfikacja kodu
  $sql=”SELECT security_code_timeout from users where
       login=’”.$_SESSION[‘login’].”’ AND
       security_code=”.$_REQUEST[“code”].
       “ AND id=”.$_SESSION[‘user_id’];
  echo $sql.”<BR/>”;
  $result = $this->conn->query($sql);
  //pobranie rekordu i zapisanie w tablicy asocjacyjnej
  $row = $result->fetch_assoc();
```

```
  //weryfikacja ważności kodu
  $now = date('Y-m-d H:i:s', time());
  if($row['security_code_timeout']>=$now){
   //pomyślnie zakończono drugi etap autentykacji
   $_SESSION['logged_in']='logged_in';
   return true;
  }
 }
 return false;
}
```

Pomyślne zakończenie logowania dwuskładnikowego za pomocą zaprezentowanego w niniejszym rozdziale kodu powoduje jedynie wyświetlenie komunikatu dla użytkownika. Mimo tego, kod jest w pełni funkcjonalny, bowiem fakt zalogowania użytkownika utrwalany jest w sesji. Jeśli chcemy wyświetlić zalogowanemu użytkownikowi dodatkową zawartość na stronie, wystarczy zweryfikować wartość komórki `logged_in` przechowywanej w tablicy sesji. Jeśli ma ona wartość 'logged_in" oznacza to, że użytkownik jest zalogowany i można mu udostępnić zawartość przeznaczoną dla użytkowników zalogowanych.

3.3. Zabezpieczenia przed atakami

Mechanizm logowania należy zabezpieczyć przed atakami polegającymi na zgadywaniu hasła. Mogą zostać użyte dwie odmiany tego ataku. Pierwsza polega na zgadywaniu hasła określonego użytkownika. Atakujący podejmuje próby zalogowania się na wybrane konto przy użyciu różnych haseł. Stara się w ten sposób odgadnąć hasło użytkownika. Atak ten możemy rozpoznać weryfikując liczbę następujących po sobie niepoprawnych logowań. Próby tego typu ataku możemy również rozpoznawać analizując liczbę wszystkich niepoprawnych prób logowań na każde z kont. Drugim typem ataku jest atak zgadywania haseł (i być może loginów) do dowolnych kont. Atak ten polega na próbach logowania się na różne konta z użyciem popularnych haseł. Charakterystyczną cechą tego ataku są pojedyncze próby logowania na poszczególne konta. Dzięki temu, mechanizmy weryfikujące występowanie pierwszego typu ataku nie zadziałają w tym przypadku. Obrona przed tego typu atakiem jest trudniejsza. Wymaga ona rozpoznania, że atakujący podejmuje wiele nieskutecznych prób logowania. Jedną z możliwości jest rejestrowanie prób logowań przychodzących z każdego z adresów IP. W przypadku wykrycia zbyt dużej liczby błędnych logowań z określonego adresu możemy nałożyć blokadę.

Blokada polega na zablokowaniu możliwości logowania się. Może ona dotyczyć wybranego użytkownika lub adresu. Istotne jest jednak, aby stosować blokady czasowe, czyli blokować możliwość logowania się przez określony czas. Raczej unikamy stosowania blokad trwałych. Zdarza się bowiem, że użytkownik zapomni hasło czy przez przypadek kilkukrotnie poda niewłaściwe. Trwałe zablokowanie konta skutkowałoby naruszeniem dostępności do aplikacji. Mechanizm ten mógłby również zostać wykorzystany przez atakujących. Wystarczyłoby kilka błędnych prób logowania na każde z kont by uczynić system niedostępnym. Dlatego też stosujemy blokady czasowe. Są one wystarczającym zabezpieczeniem. Znacząco wydłużają czas ataku, czyniąc go nieefektywnym.

Wprowadzimy teraz do naszej aplikacji omówione mechanizmy bezpieczeństwa. Zaczniemy od zabezpieczenia konta użytkownika przed zgadywaniem hasła. Naszym celem jest blokowanie konta użytkownika na okres pięciu minut, po co najmniej trzech nieudanych próbach logowania. Będziemy potrzebowali dodatkowych pól w tabeli users. Będą to:

- last_incorrect_logins – pozwalające na przechowanie licznika nieudanych prób logowania,
- temporary_blockade – służące do zapisania czasu zakończenia blokady czasowej.

Do utworzenia dodatkowych pól w bazie danych użyjemy skryptu zaprezentowanego na listingu 3.13.

Listing 3.13. Skrypt dodający pola do tabeli users

```
ALTER TABLE users
  ADD COLUMN last_incorrect_logins smallint(6)
   DEFAULT 0 NOT NULL ;
 ALTER TABLE users
  ADD COLUMN temporary_blockade timestamp ;
```

Następnie modyfikujemy istniejący proces logowania użytkownika. Do funkcji user_login_2F_step1() dodajemy sprawdzenie, czy konto użytkownika nie jest aktualnie zablokowane. Sprawdzenia tego dokonujemy przed operacją weryfikacji hasła. Jeśli blokada jest aktywna, przerywamy proces logowania i wyświetlamy użytkownikowi stosowny komunikat.

W przypadku, gdy konto nie jest zablokowane, przystępujemy do weryfikacji hasła. Jeśli hasło jest niepoprawne, zwiększamy wartość licznika niepoprawnych prób logowania (pole last_incorrect_logins). Jeśli licznik przekroczy wartość trzy, nakładamy blokadę czasową zapisując czas końca blokady w rekordzie użytkownika. Omawiane modyfikacje kodu zaprezentowano na listingu 3.14.

Listing 3.14. Modyfikacje funkcji user_login_2F_step1

```
public function user_login_2F_step1($login, $password)
 {
  ...
   //pobranie z bazy danych informacji o użytkowniku
   $sql = "SELECT id,login,password,email,
     last_incorrect_logins, temporary_blockade from users
     where login=" . $login;
   $result = $this->conn->query($sql);
   //sprawdzenie czy tylko jeden rekord został zwrócony
   if ($result->num_rows == 1) {
    //pobranie rekordu i zapisanie w tablicy asocjacyjnej
    $row = $result->fetch_assoc();
    //weryfikacja blokady tymczasowej
    $now = date('Y-m-d H:i:s', time());
    if ($row['temporary_blockade'] >= $now) {
     $answer[,communicate'] = „Blokada czasowa konta do „
      . $row[,temporary_blockade'];
     echo ,Twoje konto zostało czasowo zablokowane do ,
      .$row['temporary_blockade'].' <br/>';
     return false;
    }

    //weryfikacja hasła
    if ($this->crypto->verify_bcrypt($password,
      $row[,password'])) {
     //pomyślnie zakończono pierwszy etap autentykacji
    ...
    } else {
     //błędne hasło
     ...
     //zaktualizuj liczniki błędnych prób logowań
     $row[,last_incorrect_logins'] =
        $row[,last_incorrect_logins'] + 1;
     if ($row[,last_incorrect_logins'] > 2) {
      //nałóż blokadę czasową
      $time = date('Y-m-d H:i:s', time() + 300);
      $sql = "Update users set last_incorrect_logins="
        . $row[,last_incorrect_logins']
        . „, temporary_blockade='" . $time . „'
        where id=" . $row[,id'];
     } else {
      $sql = „Update users set last_incorrect_logins="
        . $row[,last_incorrect_logins'] . „ where id="
```

```
        . $row[,id'];
      }
      $this->conn->query($sql);
    }
    …
```

Aby omawiane zabezpieczenie działało prawidłowo konieczne jest również zerowanie licznika niepoprawnych prób logowania w momencie, gdy logowanie powiedzie się. Wprowadzimy więc modyfikację do funkcji `user_login_2F_step2()`. Po pomyślnym zakończeniu procesu logowania będziemy zerować licznik błędnych prób logowań użytkownika, tak jak to pokazano na listingu 3.15.

Listing 3.15. Modyfikacje funkcji user_login_2F_step2

```
public function user_login_2F_step2($code)
 {
  ...
   if ($row[,security_code_timeout'] >= $now) {
    //pomyślnie zakończono drugi etap autentykacji
    $_SESSION[,logged_in'] = ,logged_in';
    $sql = „update users set last_incorrect_logins=0
      where id=" . $_SESSION[,user_id'];
    $this->conn->query($sql);
    return true;
   }
  }
  return false;
 }
```

Zaimplementowane zabezpieczenie nakłada blokadę czasową na konto użytkownika po kilku niepoprawnych próbach logowania. Jednak nadal możliwe jest atakowanie aplikacji za pomocą ataku „password spray" polegającego na próbie logowania na wiele kont za pomocą wybranego hasła [22]. Aby ograniczyć możliwość tego typu ataku, dodamy do naszej aplikacji funkcjonalność nakładania blokad na adresy IP, z których następuje duża liczba prób logowań zakończonych niepowodzeniem. Będziemy do tego potrzebowali struktury do przechowywania informacji o adresach IP. W tym celu utworzymy w bazie danych tabelę ip_addresses, która będzie przechowywała dane adresów IP oraz liczby poprawnych i niepoprawnych logowań z adresu, jak również informacje o blokadach adresów. Do utworzenia tabeli użyjemy skryptu z listingu 3.16.

Listing 3.16. Utworzenie tabeli ip_addresses

```
CREATE TABLE ip_addresses (
  id bigint(20) NOT NULL AUTO_INCREMENT,
  correct_logins_num bigint(20) DEFAULT 0 NOT NULL
   comment ,liczba_poprawnych_logowan_z_adresu',
  incorrect_logins_num  int(11) DEFAULT 0 NOT NULL comment
   ,liczba_niepoprawnych_logowan_z_adresu',
  last_incorrect_logins smallint(6) DEFAULT 0 NOT NULL
   comment ,liczba_ostatnich_kolejnych_bl_logowan',
  permanent_blockade    tinyint(1) DEFAULT false NOT NULL
   comment ,czy zablokowany_na_stale',
  temporary_blockade    timestamp NULL
   comment ,czas zakończenia blokady',
  ip_address             varchar(255) NOT NULL,
  PRIMARY KEY (id));
```

Zanim podjęta zostanie próba zalogowania użytkownika, będziemy weryfikować adres IP, z którego nadeszło żądanie logowania. W tym celu utworzymy dedykowaną funkcję verify_ip(). Będzie to uniwersalna funkcja weryfikująca czy adres IP jest zablokowany oraz aktualizująca liczniki prób logowania. Funkcja ta będzie przyjmowała jako parametry: adres IP, z którego następuje próba logowania oraz wynik logowania. Wynik logowania może przyjmować wartości: 'yes' w przypadku pomyślnego zalogowania użytkownika, 'no' w przypadku błędnej próby logowania oraz 'undefined' w przypadku trwającej próby logowania, dla której wynik jeszcze nie jest znany. Funkcja weryfikuje, czy przekazany adres już istnieje w bazie danych. Jeśli nie, to dodaje odpowiedni wpis. Potem następuje sprawdzenie, czy na adres nałożono blokadę czasową lub trwałą. W przypadku zidentyfikowania jakiejkolwiek blokady funkcja zwraca negatywny wynik weryfikacji adresu IP. Oznacza to, że możliwość logowania z analizowanego adresu powinna być zablokowana.

Jeśli jednak adres nie jest zablokowany, to następuje weryfikacja przekazanej do funkcji informacji o wyniku logowania (pomyślne, nieokreślone lub błędne). W przypadku pomyślnego zakończenia logowania licznik kolejnych nieudanych prób logowania jest zerowany. W przypadku próby logowania zakończonej niepowodzeniem licznik kolejnych nieudanych prób logowania jest inkrementowany oraz następuje weryfikacja jego wartości w celu stwierdzenia, czy należy nałożyć blokadę czasową lub stałą. Nałożenie blokady następuje po przekroczeniu zdefiniowanej liczby kolejnych niepoprawnych logowań. W przypadku niezakończonej próby logowania weryfikowane są jedynie blokady. Liczniki nie są modyfikowane. Funkcja realizująca weryfikację adresu IP została przedstawiona na listingu 3.17.

Listing 3.17. Funkcja weryfikująca adres IP

```
public function verify_ip($ip, $is_login_successfull)
{
 //tablica na wartość zwracaną przez funkcję
 $answer = [
  ,result' => false, //wynik weryfikacji adresu IP
  ,communicate' => ,', //komunikat błędu
  ,ip' => „undefined" // identyfikator adresu IP
   id z bazy danych)
 ];
 //zweryfikowanie czy adres IP istnieje w bazie
 $sql = „SELECT * FROM ip_addresses WHERE ip_address='"
  . $ip . „'";
 $result = $this->conn->query($sql);
 //sprawdzenie czy rekord został zwrócony
 if (!$result->num_rows) {
  //adresu nie ma w bazie
  //wprowadź adres do bazy i pobierz jego identyfikator
  $sql = „insert into ip_addresses (incorrect_logins_num,
   last_incorrect_logins,correct_logins_num,
   permanent_blockade,ip_address ) values (0,0,0,0,'"
   . $ip . „')";
  $this->conn->query($sql);
  $sql = „SELECT * FROM ip_addresses WHERE ip_address='"
   . $ip . „'";
  $result = $this->conn->query($sql);
 }
 $row = $result->fetch_assoc();
 $answer[,ip'] = $row[,id'];
 //pobranie bieżącego czasu
 $now = date('Y-m-d H:i:s', time());
 if ($row[,temporary_blockade'] >= $now) {
  //adres jest czasowo zablokowany
  $answer[,communicate'] = „Blokada czasowa adresu IP do „
   .$row[,temporary_blockade'];
 } elseif ($row[,permanent_blockade'] == 1) {
  //adres jest zablokowany bezterminowo
  $answer[,communicate'] = „Blokada trwała adresu IP";
 } elseif ($is_login_successfull == ,yes') {
  //użytkownik zalogował się pomyślnie z analizowanego
    adresu IP
  $answer[,result'] = „true";
  //zarejestruj poprawne logowanie z adresu
  $row[,correct_logins_num'] = $row[,correct_logins_num']
```

```
  + 1;
 $sql = „Update ip_addresses set correct_logins_num=”
  . $row[‘correct_logins_num’] .
  “,last_incorrect_logins=0 where id=” . $row[,id’];
 $this->conn->query($sql);
} elseif ($is_login_successfull == ,no’) {
 //błędne logowanie z analizowanego adresu IP
 $answer[,result’] = true;
 //zarejestruj błędne logowanie z adresu
 $row[,last_incorrect_logins’] =
  $row[,last_incorrect_logins’] + 1;
 $row[,incorrect_logins_num’] =
  $row[,incorrect_logins_num’] + 1;
 if ($row[,last_incorrect_logins’] > 5) {
  //liczba kolejnych bł. logowań z adresu większa od 5
  //oblicz i zapisz w bazie czas blokady (5 minut)
  $time = date(‘Y-m-d H:i:s’, time() + 300);
  $sql = “update ip_addresses set incorrect_logins_num=”
   . $row[‘incorrect_logins_num’] .
   “, temporary_blockade=’” . $time . „’,
   last_incorrect_logins=” .
   $row[‘last_incorrect_logins’] .
   „ where id=” . $row[,id’];
 } elseif ($row[,last_incorrect_logins’] > 50) {
  //liczba kolejnych bł. logowań z adresu większa od 50
  //nałóż blokadę stałą
  $sql = „update ip_addresses set incorrect_logins_num=”
   . $row[,incorrect_logins_num’] .
   „, permanent_blockade=1, last_incorrect_logins=”
   . $row[‘last_incorrect_logins’] .
   “ where id=” . $row[,id’];
 } else {
  //liczba kolejnych bł. logowań z adresu mniejsza niż 4
  //zwiększ licznik błędnych logowań z adresu
  $sql = „update ip_addresses set incorrect_logins_num=”
   . $row[,incorrect_logins_num’]
   . „, last_incorrect_logins=”
   . $row[,last_incorrect_logins’]
   . „ where id=” . $row[,id’];
 }
 echo „<BR/>” . $sql . „<BR/>”;
 $this->conn->query($sql);
} else {
 //logowanie niezakończone (w trakcie)
 $answer[,result’] = true;
```

```
  }
  return $answer;
}
```

Funkcję weryfikującą adresy IP należy wywoływać w procesie logowania użytkownika. Wywołujemy ją po raz pierwszy w momencie rozpoczęcia obsługi żądania logowania. Weryfikuje ona wtedy, czy adres IP nie jest zablokowany. W przypadku wykrycia blokady proces logowania jest przerywany, a użytkownikowi wyświetlany jest stosowny komunikat. Funkcja ta powinna zostać również wywołana w przypadku podania błędnego loginu, hasła lub kodu jednorazowego. Posłuży ona wtedy do zarejestrowania nieudanej próby logowania i ewentualnego nałożenia blokady na adres IP. W naszej przykładowej aplikacji wszystkie wymienione wywołania umieszczone powinny zostać wewnątrz funkcji user_login_2F_step1() lub user_login_2F_step2(). Tak, jak to zaprezentowano na listingach 3.18 i 3.19. Ostatnie wywołanie omawianej funkcji powinno zostać umieszczone w funkcji user_login_2F_step2() po pomyślnym zakończeniu procesu logowania. Wyzeruje ona wówczas licznik kolejnych nieudanych prób logowania.

Listing 3.18. Wywołania funkcji verify_ip() wewnątrz funkcji user_login_2F_step1()

```
public function user_login_2F_step1($login, $password)
{
 //zweryfikowanie adresu IP
 $ip = $_SERVER[,REMOTE_ADDR’];
 $answer = $this->verify_ip($ip, ‘undefined’);
 if (!$answer[,result’]) {
  echo $answer[,communicate’];
 } else {
  //...
  //sprawdzenie czy tylko jeden rekord został zwrócony
  if ($result->num_rows == 1) {
   ...

   //weryfikacja hasła
   if ($this->crypto->verify_bcrypt($password,
    $row[,password’])) {
    //pomyślnie zakończono pierwszy etap autentykacji
    ...
   } else {
    //błędne hasło
    $ip = $_SERVER[,REMOTE_ADDR’];
    //$row = $result->fetch_assoc();
    $uid = $row[,id’];
    $answer = $this->verify_ip($ip, ‘no’);
```

```
    ...

    }
   } else {
    //login nie istnieje w bazie
    $ip = $_SERVER[,REMOTE_ADDR'];
    $answer = $this->verify_ip($ip, 'no');
    $this->user_login_register($login, $answer['ip'],
     false, false);
   }
  }
  return false;
 }
```

Listing 3.19. Wywołania funkcji verify_ip() wewnątrz funkcji user_login_2F_step2()

```
public function user_login_2F_step2($code)
 {
  //zweryfikowanie czy użytkownik pomyślnie ukończył
   pierwszy etap logowania
  if ($_SESSION[,logged_in'] == „after_first_step") {
   $code = „'" . $code . „'";
   //pobranie z bazy danych informacji o użytkowniku
    i weryfikacja kodu
   $sql = „SELECT security_code_timeout from users where
    login='" . $_SESSION[,login'] . „' AND security_code="
    . $_REQUEST[„code"] ." AND id=" . $_SESSION[,user_id'];
   echo $sql . „<BR/>";
   $result = $this->conn->query($sql);
   if ($result->num_rows == 0) {
    $ip = $_SERVER[,REMOTE_ADDR'];
    $this->verify_ip($ip, 'no');
   }

   ...
   if ($row[,security_code_timeout'] >= $now) {
    //pomyślnie zakończono drugi etap autentykacji
    $_SESSION[,logged_in'] = ,logged_in';
    $ip = $_SERVER[,REMOTE_ADDR'];
    $answer = $this->verify_ip($ip, 'yes');
    ...
    return true;
   }
  }
  return false;
 }
```

3.4. Zadanie do samodzielnego wykonania

Aby jeszcze skuteczniej identyfikować atakujących, spróbuj rozpoznawać komputery, z których korzystają użytkownicy. Na komputerze każdego z użytkowników umieść w ciasteczku niepowtarzalny identyfikator. Dodaj w bazie danych tabelę computers, w której przechowasz dane komputerów. Komputery, z których wielokrotnie ktoś poprawnie się logował traktuj jako zaufane. Komputery, z których wykonano wiele nieudanych prób logowania blokuj na 10 minut po każdej nieudanej próbie logowania.

4. Wstrzykiwanie kodu

Wstrzykiwanie kodu jest atakiem na aplikacje internetowe polegającym na wprowadzaniu do aplikacji dodatkowych instrukcji. Dokonuje się tego najczęściej poprzez wprowadzenie spreparowanej zawartości do pól formularza. Atakujący najczęściej wstrzykują kod SQL, JavaScript lub HTML[3]. Według rankingu zagrożeń tworzonego przez organizację OWASP ataki wstrzyknięcia (ang. injection) zajęły w 2021 roku trzecie miejsce. Pokazuje to, że ataki tego typu są groźne i na dodatek często stosowane.

4.1. Wstrzykiwanie kodu SQL

Wstrzykiwanie kodu SQL jest techniką atakowania aplikacji bazodanowych polegającą na dołączaniu dodatkowych instrukcji do poleceń SQL wbudowanych w aplikację. Tego typu ataki stosuje się przeciwko aplikacjom internetowym. Polegają one na wprowadzeniu do formularza odpowiednio spreparowanych fragmentów kodu SQL, które powodują niepoprawne działanie aplikacji [1].

W niniejszym rozdziale zaprezentowane zostaną wybrane techniki ataków polegających na wstrzykiwaniu kodu SQL oraz niektóre z możliwych do uzyskania efektów. Prezentacja technik ataku ma na celu wspomożenie użytkowników w zrozumieniu omawianego typu ataku oraz dostarczeniu wiedzy na temat sposobu weryfikowania podatności aplikacji na ataki wstrzykiwania kodu SQL.

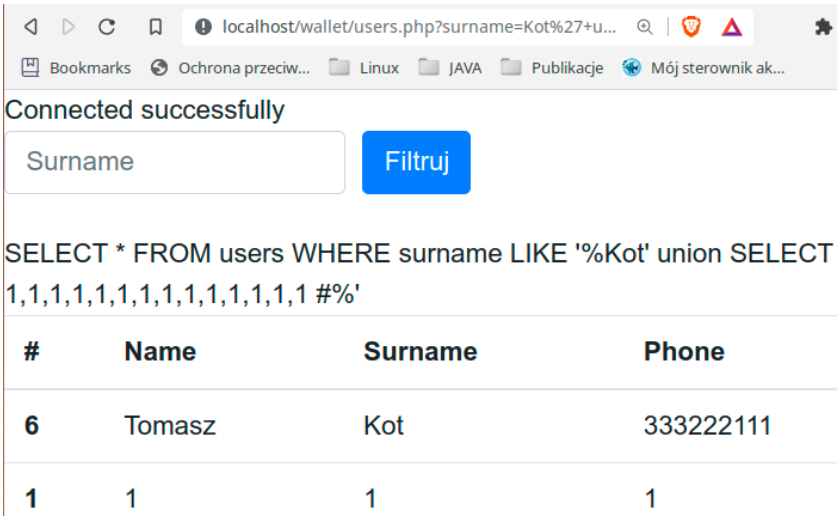
Celem atakującego jest najczęściej pozyskanie danych z bazy bez posiadania odpowiednich uprawnień. Wykorzystuje on więc techniki modyfikacji zapytań, które mają na celu ujawnienie dodatkowych danych. Często spotykane jest użycie:

- instrukcji `union`, która pozwala na dołączenie do wyników istniejącego zapytania wyniku zwracanego przez zapytanie dołączone przez atakującego,
- tautologii, będącej wyrażeniem zawsze prawdziwym, pozwalającej na modyfikowanie warunków podawanych po słowie kluczowym `where`,
- znaku komentarza pozwalającego na zakomentowanie części zapytania.

Przykład ataku wstrzyknięcia kodu SQL przedstawimy analizując możliwość przeprowadzenia ataku na stronę users.php naszej aplikacji. Strona ta wyświetla listę użytkowników oferując jednocześnie możliwość filtrowania wyników poprzez podanie w formularzu ciągu znaków, jakie mają się znaleźć w wyświetlonych nazwiskach. Formularza tego użyjemy do wprowadzenia kodu SQL, który spowoduje wyświetlenie danych z tabeli passwords. Użyjemy do tego instrukcji `union` oraz dodatkowego zapytania, którego wyniki dołączymy do wyświetlanych na stronie.

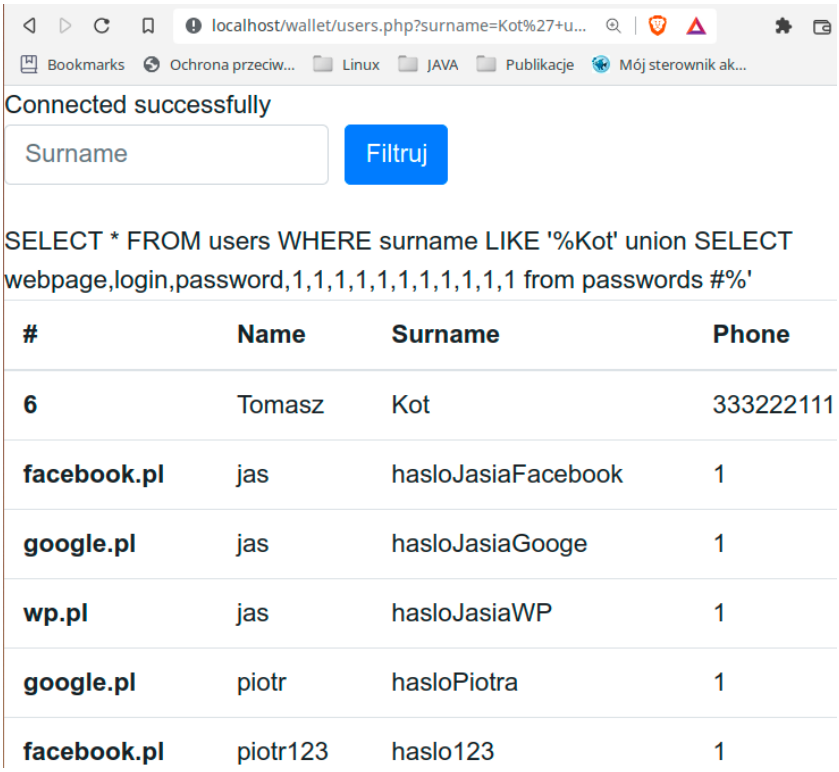
Instrukcja union połączy wyniki dwóch zapytań tylko wtedy, gdy liczba kolumn zwracanych przez obydwa z nich jest taka sama. Atak zaczniemy od zweryfikowania jaką liczbę kolumn zwraca zapytanie, które jest wykorzystywane przez funkcję filtrowania.

Do formularza wpisujemy ciąg znaków, który ma zostać użyty do filtrowania wyświetlanych nazwisk. Ciąg ten kończymy znakiem apostrofu. Znak ten zostanie zinterpretowany jako koniec danych pobranych z formularza. Dzięki temu kolejne wprowadzone do formularza ciągi znaków będą już traktowane jako dalsza część polecenia SQL. Dlatego też w formularzu umieszczamy dodatkowo słowo `union` a po nim instrukcję `select`, która zwraca listę dowolnych wartości. Jeśli liczba elementów na liście jest taka sama jak liczba kolumn zwracanych przez zapytanie używane w aplikacji, wówczas na stronie powinny zostać wyświetlone wyniki – tak jak to zaprezentowano na rysunku 4.1. Jeśli jednak liczba zwróconych wartości będzie niewłaściwa, wówczas nie zobaczymy żadnych rezultatów wyszukiwania. Oznacza to, że należy zmodyfikować liczbę elementów listy. Na rysunku 4.1 przedstawiono wynik uzyskany po wstrzyknięciu do formularza ciągu „Kot' union SELECT 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1 #". Ciąg ten zakończony jest znakiem komentarza. Znak ten jest niezbędny, aby „ukryć" przed interpreterem języka SQL dalszą część polecenia SQL, które jest wbudowane w aplikację. Bez tego polecenie miałoby błędną składnię i nie udałoby się uzyskać żadnych wyników. Zapytanie, które zostało wykonane podczas ataku widoczne jest na prezentowanym zrzucie ekranu.



Rys. 4.1. Wyniki wstrzyknięcia kodu SQL

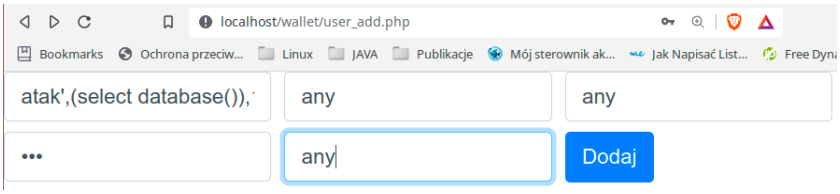
Po określeniu liczby kolumn, jaką powinno zwracać wstrzyknięte zapytanie, możemy przystąpić do właściwego ataku. Zmodyfikujemy wstrzykiwany kod SQL tak by zwracał dane z tabeli passwords. Wprowadzimy do formularza ciąg: „Kot' union SELECT webpage,login,password,1,1,1,1,1,1,1,1,1,1,1 from passwords #". Efekt ataku przedstawiony został na rysunku 4.2. Widzimy tam dane logowania do różnych witryn internetowych. Co prawda wynik wygląda nieco nieporządnie – nagłówki kolumn zawierają oryginalne nazwy a kolumna „Phone" wypełniona jest jedynkami. Pamiętajmy jednak, że celem ataku było uzyskanie dostępu do danych a nie wyświetlenie ich w elegancki sposób. Atakujący jest świadomy uzyskanego wyniku i potrafi go odczytać.



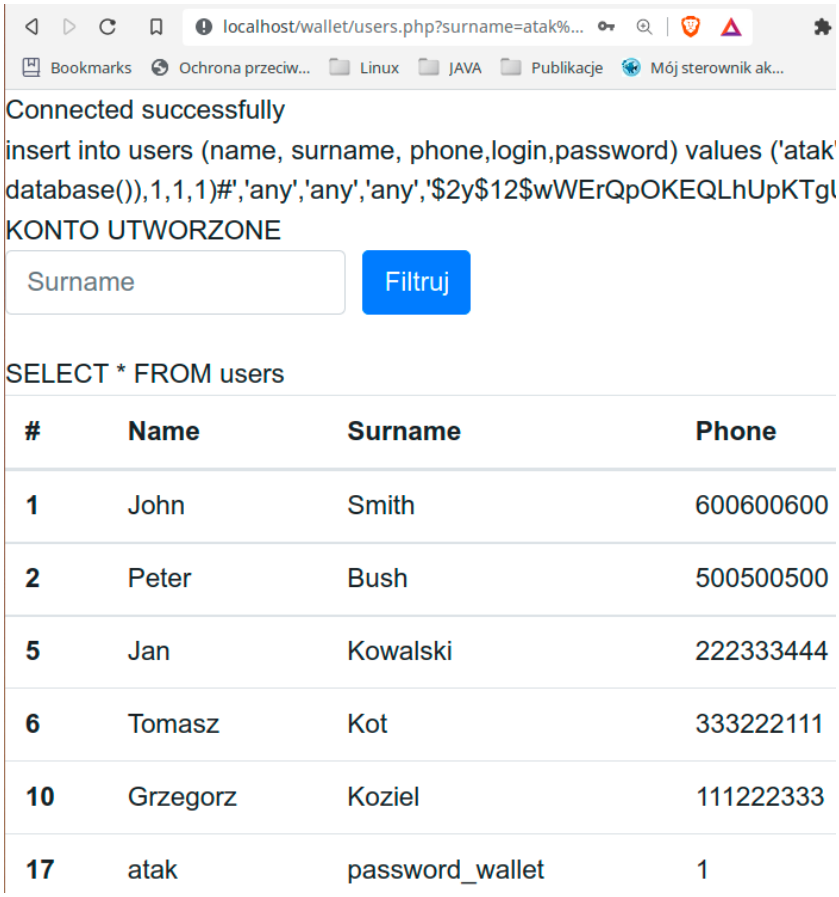
Rys. 4.2. Dane uzyskane w wyniku wstrzyknięcia kodu SQL

Nie tylko instrukcja `select` jest wrażliwa na ataki. Każda inna instrukcja języka SQL może stanowić furtkę dla atakującego. Zaprezentujemy to na przykładzie instrukcji `insert`. Nasza aplikacja udostępnia możliwość dodania nowego konta użytkownika. Dokonujemy tego poprzez wypełnienie pól formularza na stronie user_add.php. Jeśli jednak zamiast danych użytkownika wprowadzimy

do formularza kod SQL zwracający dane, zostaną one zapisane w bazie danych. Następnie możliwe będzie wyświetlenie tychże danych w interfejsie użytkownika. Sposób i efekt przeprowadzenia tego typu ataku, polegającego na wprowadzeniu do pola imienia ciągu „atak',(select database()),1,1,1)#", został zaprezentowany na rysunkach 4.3 i 4.4. W wyniku ataku zostało utworzone nowe konto użytkownika, a w polu nazwiska umieszczona została nazwa bazy danych używanej przez atakowaną aplikację.



Rys. 4.3. Atak wstrzyknięcia kodu SQL do formularza dodawania użytkownika



#	Name	Surname	Phone
1	John	Smith	600600600
2	Peter	Bush	500500500
5	Jan	Kowalski	222333444
6	Tomasz	Kot	333222111
10	Grzegorz	Koziel	111222333
17	atak	password_wallet	1

Rys. 4.4. Efekt ataku wstrzyknięcia kodu SQL do formularza dodawania użytkownika

Zaprezentowane ataki jedynie pokazują zasadę działania ataków wstrzykiwania kodu SQL. Nie wyczerpują tematu. Istnieje bowiem o wiele więcej możliwości wykonania tego typu ataku.

4.2. Zabezpieczenia przed wstrzykiwaniem kodu SQL

Istnieje wiele technik zabezpieczania przed atakami wstrzykiwania kodu SQL. Najważniejsze z nich to:

- użycie szablonów zapytań,
- wykorzystanie bezpiecznej biblioteki do operacji bazodanowych,
- filtrowanie i walidacja danych,
- białe i czarne listy.

4.2.1. Szablony zapytań

Zapytania, które dotychczas konstruowaliśmy, składały się z poleceń języka SQL oraz danych. Łańcuch znaków zawierający polecenie połączone z danymi był przesyłany do silnika bazodanowego, który parsował je, czyli rozpoznawał składniki tego polecenia określając, czy stanowią one treść polecenia czy też dane. W przypadku wstrzyknięcia odpowiednio spreparowanego ciągu, silnik bazy danych mógł zostać oszukany i potraktować część danych otrzymanych od użytkownika jako treść polecenia SQL. Naszym zamiarem jest uniemożliwienie oszukiwania serwera bazodanowego. Chcemy by był on w stanie jednoznacznie odróżnić polecenie SQL od dołączonych do niego danych, niezależnie od ich treści. Taką możliwość dają nam szablony zapytań. Szablon zapytania to polecenie SQL nie zawierające danych. Miejsca, w których powinny zostać umieszczone dane są oznaczane za pomocą znaków zapytania lub etykiet. Przykładowy szablon zapytania ma postać: `select * from users where email=?`. Tak przygotowany szablon wysyłany jest do serwera bazodanowego, który parsuje, kompiluje i optymalizuje otrzymane zapytanie. Tak przygotowane zapytanie jest składowane w pamięci serwera. W kolejnym etapie z szablonem zapytania wiązane są dane. Oznacza to, że w oddzielnym pakiecie przesyłane są do serwera wartości, które powinny zostać wstawione do szablonu zapytania. Jako, że przesyłane są one oddzielnie, serwer może je jednoznacznie rozpoznać i nie potraktuje ich jako fragmentu zapytania SQL [17].

Użycie szablonu zapytań zaprezentowane zostanie na przykładzie strony users.php. Dotychczas lista użytkowników pobierana była z serwera bazodanowego za pomocą funkcji `select()`, która jako parametr pobierała kompletne zapytanie SQL zawierające w swojej treści dane. Funkcję tę zastąpimy funkcją `get_users()`,

do której jako parametr przekażemy łańcuch znaków pobrany z formularza wyszukiwania. Funkcja zweryfikuje długość otrzymanego łańcucha znaków. Jeśli zawiera on jakieś znaki oznacza to, że użytkownik podał łańcuch znaków do wyszukiwania i należy uwzględnić to w zapytaniu SQL. Jeśli łańcuch znaków ma zerową długość to znaczy, że nie podano żadnego kryterium wyszukiwania i należy pobrać rekordy wszystkich użytkowników.

Jeśli do funkcji przekazany został parametr, wówczas otrzymany ciąg znaków umieszczamy pomiędzy znakami „%" by umożliwić wyszukanie podanego ciągu niezależnie od jego położenia (na początku, w środku czy na końcu nazwiska). Następnie konstruujemy szablon zapytania i wysyłamy go do serwera bazodanowego. Z gotowym szablonem wiążemy dane, przesyłając je w oddzielnym pakiecie. Następnie możemy wydać polecenie wykonania zapytania oraz pobrać wyniki.

Szablon zapytania pozostaje aktywny i gotowy do użycia dopóki nie zostanie wydane polecenie jego dealokacji. Pozwala to na wielokrotne użycie tego samego szablonu poprzez wiązanie go z różnymi zestawami danych i wykonywanie. Oprócz podniesienia poziomu bezpieczeństwa, pozwala to na uzyskanie większej prędkości działania aplikacji – nie ma bowiem konieczności wielokrotnego parsowania i optymalizowania tego samego kodu SQL.

W przypadku wykonywania zapytania bez parametrów możemy również posłużyć się szablonem zapytania. Pomijamy wówczas instrukcję wiązania szablonu z danymi. Kod funkcji `get_users()`, która zwraca zestaw danych dla strony users.php został zaprezentowany na listingu 4.1.

Listing 4.1. Kod funkcji get_users()

```
public function get_users($surname)
 {
  //weryfikacja czy w otrzymanym łańcuchu są jakieś znaki
  if(strlen($surname)>0){
   //otrzymano łańcuch znaków do wyszukania
   $surname='%'.$surname.'%';
   //skonstruowanie szablonu zapytania
   $sql = "SELECT * FROM users WHERE surname LIKE ?";
   $stmt = $this->conn->prepare($sql);
   //wiązanie danych z szablonem
   $stmt->bind_param('s',$surname);
   //wykonanie zapytania
   $stmt->execute();
  }else{
   //brak parametru - wykonanie zapytania bez parametrów
   //przygotowanie szablonu
   $sql = „SELECT * FROM users";
```

```
    $stmt = $this->conn->prepare($sql);
    //wykonanie zapytania
    $stmt->execute();
   }
   //pobranie wyników
   $result = $stmt->get_result();
   //dealokacja szablonu
   $stmt->close();
   //zwrócenie wyników zapytania
   return $result;
  }
```

4.2.2. PDO

Biblioteka PDO (ang. PHP Data Objects) jest rozwiązaniem zapewniającym uniwersalny interfejs pozwalający na uzyskanie dostępu do wielu silników baz danych. Posiada wbudowane sterowniki do takich baz jak MySQL, PostgreSQL, Oracle, SQLite, Firebird, IBM oraz innych [18]. Biblioteka ta w naturalny sposób implementuje szablony zapytań, pozwalając uniknąć ręcznej implementacji szczegółów technicznych, które mogą się różnić w zależności od zastosowanego silnika bazy danych.

Połączenie z bazą danych nawiązywane jest przez konstruktor klasy PDO, do którego jako parametry przekazujemy:

- DSN (ang. Data Source Name) – nazwę źródła danych zawierającą adres serwera bazodanowego oraz nazwę bazy danych,
- nazwę użytkownika,
- hasło użytkownika.

Używając klasy PDO należy zwrócić uwagę na prawidłowe obsłużenie wyjątków. Klasa ta szczegółowo raportuje błędy i wyjątki, często zawierając w komunikatach dane wrażliwe. Wywołanie konstruktora klasy PDO należy umieścić wewnątrz bloku obsługi wyjątków. W przypadku wygenerowania wyjątku należy zwrócić uwagę na to, by komunikat przekazywany w obiekcie wyjątku nie był dostępny dla użytkowników. Może on zawierać wrażliwe dane, takie jak login i hasło użytkownika bazodanowego. Dobrym rozwiązaniem jest zapisywanie szczegółowych komunikatów do pliku. Użytkownikowi natomiast powinny zostać wyświetlone ogólne informacje w postaci komunikatów przygotowanych przez programistę.

Obsługa zapytań w PDO przebiega analogicznie do realizowanej za pomocą szablonów zapytań – jest to bowiem ten sam mechanizm. Przygotowany szablon zapytania należy przesłać do serwera bazodanowego za pomocą funkcji `prepare()` a następnie powiązać z danymi funkcją `bindValue()`. Wykonanie zapytania odbywa się za pomocą funkcji `execute()`. Wyniki zostają umieszczone

w obiekcie reprezentującym szablon zapytania i mogą zostać pobrane wiersz po wierszu za pomocą funkcji `fetch()`.

Aby zaprezentować użycie biblioteki PDO podmienimy mechanizm pobierania listy użytkowników z bazy danych na nowy, bazujący na bibliotece PDO. W tym celu utworzymy klasę *DB_PDO*, która będzie się komunikowała z bazą danych przy użyciu biblioteki PDO. Klasa ta będzie posiadała konstruktor odpowiedzialny za nawiązanie połączenia. Umieścimy w nim dane niezbędne do nawiązania połączenia oraz wywołamy konstruktor klasy PDO nawiązujący połączenie.

Do klasy dodamy metodę `get_users()`, która będzie odpowiedzialna za pobranie listy użytkowników z bazy danych. Metoda ta będzie pobierała jako parametr poszukiwany łańcuch znaków lub ciąg pusty w przypadku, gdy użytkownik nie zdefiniuje kryterium filtrowania. Następnie przygotowany zostanie szablon zapytania, z którym związane zostaną dane. Efektem działania funkcji będzie zwrócenie wyników wykonania zapytania. Kod klasy *DB_PDO* został zaprezentowany na listingu 4.2.

Listing 4.2. Kod klasy DB_PDO

```
<?php

class DB_PDO
{
 private $db;

 //konstruktor nawiązujący połączenie
 public function __construct()
 {
  //dane połączenia bazą
  $servername = „localhost”;
  $username = „root”;
  $password = „”;
  $dbname = „password_wallet”;
  $dsn = “mysql:host=” . $servername . “;dbname=”
   . $dbname;
  try {
   //wywołanie konstruktora, nawiązanie połączenia z bazą
   $this->db = new PDO($dsn, $username, $password);
  } catch (PDOException $e) {
   //obsługa błędów
   print „Błąd połączenia z bazą!: <br/>”;
   die();
  }
 }
```

```
 //pobranie listy użytkowników
 public function get_users($surname)
 {
  try {
   //weryfikacja czy w otrzymanym łańcuchu są jakieś znaki
   if (strlen($surname) > 0) {
    //otrzymano łańcuch znaków do wyszukania
    $surname = „%” . $surname . „%”;
    //skonstruowanie szablonu zapytania
    $sql = “SELECT * FROM users WHERE surname LIKE ?”;
    $stmt = $this->db->prepare($sql);
    //wiązanie danych z szablonem
    $stmt->bindValue(1, $surname, PDO::PARAM_STR);
   } else {
    //brak parametru - wykonanie zapytania bez parametrów
    //przygotowanie szablonu
    $sql = „SELECT * FROM users”;
    $stmt = $this->db->prepare($sql);
   }
   //wykonanie zapytania
   $stmt->execute();
   //zwrócenie wyników zapytania
   return $stmt;
  } catch (PDOException $e) {
   //obsługa błędów
   print „Błąd komunikacji z bazą!<br/>”;
   die();
  }
 }
}
```

Użycie klasy *DB_PDO* wymaga zmodyfikowania kodu na stronie users.php. Przede wszystkim należy dołączyć do niego plik zawierający klasę *DB_PDO*. Ponadto określenie, czy wyświetlać tabelę z listą użytkowników czy też komunikat, będzie się odbywało na podstawie sprawdzenia, czy funkcja `get_users()` zwróciła co najmniej jeden rekord. Skorzystamy więc z funkcji `fetch()`, która to pobierze rekord ze zwróconego zestawu wyników i zwróci wartość *true* lub zwróci wartość *false*, jeśli nie było dostępnych rekordów. Jako, że funkcja `fetch()` pobiera rekord i nie zapamiętuje go, to musimy zapamiętać pobrany rekord w zmiennej.

Do wyświetlenia tabeli z wynikami zastosujemy pętlę `do - while`, która to będzie pobierała kolejny rekord dopiero po zakończeniu wykonywania kodu umieszczonego w pętli. Kod strony users.php korzystającej z klasy *DB_PDO* został zaprezentowany na listingu 4.3.

Listing 4.3. Kod strony users.php

```
<?php

include_once „classes/db_pdo.php”;
include_once „classes/crypto.php”;
$db = new DB_PDO();
...
 <?php
// if (isset($_REQUEST[„filter”])){
 $surname = $_REQUEST[„surname”];
 $stmt = $db->get_users($surname);
 if ($row = $stmt->fetch()) {
  //zwrócono rekordy – wyświetlamy tabelę
 ?>
    <table class=”table”>
        <thead>
        <tr>
            <th scope=”col”>#</th>
            <th scope=”col”>Name</th>
            <th scope=”col”>Surname</th>
            <th scope=”col”>Phone</th>
        </tr>
        </thead>
        <tbody>
  <?php
  // wyświetlenie danych z wierszy tabeli
  do {
   ?>
         <tr>
             <th scope=”row”><?php echo $row[“id”] ?></th>
             <td><?php echo $row[“name”] ?></td>
             <td><?php echo $row[“surname”] ?></td>
             <td><?php echo $row[“phone”] ?></td>
         </tr>
   <?php
  } while($row = $stmt->fetch()); //weryfikujemy czy jest
     kolejny rekord do pobrania
  }
  ?>
...
</html>
```

4.2.3. Filtrowanie danych

Każde dane, jakie wbudowujemy w zapytanie powinny być traktowane jako niezaufane. Oprócz zastosowania szablonów zapytań do komunikacji z silnikiem bazy danych, wszystkie dane powinny zostać zweryfikowane przed użyciem. Oznacza to, że powinniśmy sprawdzić czy dane te nie zawierają niebezpiecznej zawartości. W tym celu użyjemy trzech rozwiązań: białych i czarnych list oraz filtrowania.

Białe listy są zbiorami wartości, które są zawsze uznawane za poprawne. Stosujemy je najczęściej wtedy, gdy mamy do czynienia z zamkniętą listą wartości – na przykład nazw dni tygodnia. Umieszczamy wówczas wszystkie poprawne nazwy na białej liście. Jeśli nazwa podana przez użytkownika występuje na zdefiniowanej liście, wówczas jest ona akceptowana i może być wbudowana w zapytanie. Białych list używamy również do określenia poprawnych wartości, które mogłyby zostać odrzucone podczas filtrowania.

Czarne listy są zbiorami ciągów znaków, których wystąpienie w danych filtrowanych jest niedozwolone. Najczęściej umieszczamy na nich instrukcje używane podczas ataków wstrzyknięcia kodu SQL. Zidentyfikowanie takiej instrukcji wewnątrz ciągu podanego przez użytkownika dyskwalifikuje go i powoduje jego odrzucenie.

Filtrowanie służy do zweryfikowania czy analizowane dane spełniają określone reguły. Często sprawdzeniu podlega typ danych czy też ich zawartość. Możliwe jest tu wykorzystanie predefiniowanych filtrów. Obszerną ich listę znajdziemy na stronie [19]. Jeśli jednak na liście nie widnieje filtr, który jest potrzebny, możemy go zdefiniować samodzielnie używając wyrażeń regularnych. Wyrażenie regularne jest wzorcem, do którego porównujemy filtrowany ciąg danych. Tutorial o tworzeniu wyrażeń regularnych znajdziesz na stronie [20].

Oprócz wymienionych technik używamy również technik walidacji danych. Najczęściej sprowadzają się one do sprawdzenia, czy użytkownik podał wszystkie wymagane dane oraz czy ich format jest właściwy.

Aby kompleksowo przefiltrować dane powinniśmy użyć wszystkich wymienionych technik. Najpierw należy przeprowadzić walidację danych pod kątem kompletności i poprawności formatu. Następnie powinna zostać użyta biała lista. Jeśli dane podane przez użytkownika znajdują się na białej liście, wówczas należy zakończyć proces filtrowania i zaakceptować podaną przez użytkownika wartość. W przeciwnym razie należy zweryfikować, czy analizowany ciąg nie zawiera elementów umieszczonych na czarnej liście. Jeśli je zidentyfikujemy, wówczas odrzucamy przetwarzany ciąg. W przeciwnym przypadku przystępujemy do filtrowania danych. Po pomyślnym zakończeniu filtrowania uznajemy dane za poprawne.

Omawiając temat filtrowania i walidacji danych musimy wspomnieć, że techniki te są używane zarówno po stronie serwera, jak i klienta. Wynika z tego, że

dane sprawdzamy dwa razy – raz w przeglądarce klienta, a drugi raz na serwerze. Weryfikacja danych po stronie klienta ma za zadanie odrzucenie błędnych zestawów danych wprowadzonych przez użytkownika. Jest to tak zwana kontrola poprawności. Zaletą tego rozwiązania jest obciążenie wykonywanymi operacjami komputera klienta. Nie musimy angażować do tego serwera, w związku z czym odciążamy go. Dzięki temu do serwera nie trafiają nieprawidłowe żądania spowodowane błędem użytkownika. Problemem przedstawionego rozwiązania jest to, że użytkownik jest w stanie dezaktywować lub ominąć ten typ weryfikacji. Dlatego też weryfikację danych wykonujemy ponownie po stronie serwera. Pomimo tego, że duża część operacji sprawdzania zapewne będzie powtórzona na serwerze nie możemy z nich zrezygnować. To one stanowią gwarancję, że dane celowo zmodyfikowane przez atakującego zostaną prawidłowo odfiltrowane.

Sposób filtrowania danych zaprezentowany zostanie na przykładzie formularza wyszukiwania użytkowników po nazwisku, znajdującego się na stronie users.php.

Aby zachować czytelność kodu funkcje filtrujące zostaną wydzielone do niezależnej klasy o nazwie *Filters*. Wewnątrz tej klasy, na potrzeby filtrowania danych pochodzących z formularza wyszukiwania użytkowników, utworzona zostanie funkcja `filterSearchNameString()`. Funkcja ta będzie przyjmowała jako parametr łańcuch znaków podany przez użytkownika i będzie zwracała wartość logiczną określającą, czy analizowany łańcuch spełnia reguły filtrowania, czy nie.

Aby uniknąć niejednoznaczności, przed przystąpieniem do filtrowania wszystkie znaki w analizowanym tekście zostaną zamienione na małe litery. Następnie zastosowana zostanie biała lista. Moglibyśmy się pokusić o umieszczenie w niej wszystkich nazwisk, jednak lista byłaby zbyt obszerna. Wykorzystamy ją więc do umieszczenia tylko niektórych nazwisk, które są poprawne, ale ze względu na reguły filtrowania mogłyby zostać odrzucone. Jak możemy zauważyć w kodzie przedstawionym na listingu 4.4, użyta tam biała lista obejmuje nazwiska zawierające ciąg znaków „from" będący ciągiem umieszczonym na czarnej liście. Czarna lista została użyta do zdefiniowania słów kluczowych używanych często podczas ataków wstrzyknięcia kodu SQL. Jeśli ciąg znaków podany przez użytkownika zawiera element przechowywany na czarnej liście, wówczas funkcja filtrująca zwraca wartość *false*.

Ostatnim mechanizmem jest filtrowanie poprzez weryfikację, czy analizowany ciąg pasuje do zdefiniowanego wyrażenia regularnego. Użyte wyrażenie określa, że filtrowany ciąg może zawierać tylko litery, spację i znak myślnika, a jego długość nie może przekroczyć dwudziestu znaków.

Listing 4.4. Klasa Filters

```
<?php
class Filters
```

```
{
 public function filterSearchNameString($name){
  //zwraca true jeśli filtrowanie powiedzie się
  //lub false w przypadku wykrycia treści niepasujących do
   zasad filtrowania

  //zamień wszystkie znaki na małe litery
  $name=strtolower($name);
  //biała lista
  $whiteList=array(„fromer”,”frommer”,”fromiński”);
  if(in_array( $name, $whiteList )){
   return true;
  }
  //czarna lista
  $blackList=array(“union”,”select”,”where”,”from”,”join”,
  „#”,”1=1”);
  //weryfikacja czy filtrowana wartość zawiera ciąg z
   czarnej listy
  foreach ($blackList as $forbidden) {
    if (strpos($name, $forbidden) !== FALSE) {
 return false;
   }
  }
  //filtrowanie
  //ciąg może zawierać tylko litery, myślniki i spacje
   i mieć 1-20 znaków
  if(filter_var($name, FILTER_VALIDATE_REGEXP,
   array(„options”=>array(„regexp”=>
    „/^[a-z, ,-]{1,20}$/”)))){
   return true;
  }
  else{
   return false;
  }
 }
}
```

Klasę „Filters” należy dołączyć do pliku users.php i stworzyć obiekt tejże klasy. Następnie, używając metody `filterSearchNameString()`, należy przefiltrować dane pobrane z formularza. Jeśli funkcja filtrująca zwróci wartość *true* oznacza to, że można bezpiecznie użyć podanego ciągu w zapytaniu. W przeciwnym wypadku należy odrzucić podany ciąg i przekazać pusty łańcuch znaków do funkcji `get_users()` tak, jak to zaprezentowano na listingu 4.5.

Listing 4.5. Strona users.php

```
<?php
...
include_once „classes/filters.php”;
...
$filter=new Filters();
...
 $surname = $_REQUEST[„surname”];
    if(strlen($surname))
    if(!$filter->filterSearchNameString($surname))
    {
     echo „Niedozwolone znaki w wyszukiwanym łańcuchu”;
     $surname=””;
    }
 $stmt = $db->get_users($surname);
 if ($row = $stmt->fetch()) {
 //zwrócono rekordy – wyświetlamy tabelę
 ...
```

4.3. Uprawnienia w bazie danych

Wykonywanie operacji na danych i strukturach danych (np. tabelach czy indeksach) przechowywanych w każdej bazie danych wymaga posiadania konta użytkownika tej bazy oraz odpowiednich uprawnień do wykonania tych operacji. Jedynym i domyślnym użytkownikiem bazy danych MySQL, bezpośrednio po jej zainstalowaniu, jest jej administrator, którego konto ma nazwę *root*. Uruchamiając program *shell* w oknie XAMPP i wydając z linii komend następujące polecenie:

```
mysql –u root
```

możemy zalogować się do bazy bez konieczności podawania hasła, co należy uznać za bardzo niebezpieczną sytuację ze względu na wysokie zagrożenie penetracją wszystkich baz danych dostępnych na danym serwerze. Dlatego też pierwszą komendą po zalogowaniu się takiego użytkownika powinna być zmiana hasła przy użyciu polecenia `SET PASSWORD`. Przykład jego użycia wygląda następująco:

```
set password = password(‚pollub%2022’);
```

Ze względów bezpieczeństwa, zaleca się, by zarządzanie użytkownikami bazy danych i ich uprawnieniami odbywało się wyłącznie z poziomu konta *root*. Ponadto, dobrą praktyką jest tworzenie kont użytkowników w taki sposób, aby

ograniczyć możliwość logowania się do bazy danych użytkownika o tej samej nazwie z dowolnego miejsca. Podczas tworzenia konta użytkownika w bazie danych istnieje możliwość wskazania adresu IP lub nazwy hosta, z którego będzie można uzyskać dostęp do takiej bazy. Dodatkowo, w celu podniesienia poziomu bezpieczeństwa dostępu do bazy danych, należy zdefiniować odrębne hasła dla tego samego konta dla połączeń przychodzących z poszczególnych hostów. Przykład instrukcji umożliwiających utworzenie konta o nazwie *pollub*, umożliwiającego dostęp do bazy danych z trzech wybranych hostów, oraz instrukcji definiujących hasła dla poszczególnych połączeń przedstawiono na listingu 4.6.

Listing 4.6. Tworzenie użytkowników bazy danych

```
create user 'pollub'@'212.182.18.39' ;
create user 'pollub'@'212.182.16.104' ;
create user 'pollub'@'pluton.pol.lublin.pl' ;
create user ,piotrm'@'localhost' identified by ,hDsBi2022';
create user ,piotrm'@'127.0.0.1' identified by ,hDsBi2022';
set password for ,pollub'@'212.182.18.39' = PASSWORD(,cs%20
22');
set password for ,pollub'@'212.182.16.104' =
PASSWORD(,weii%2022');
set password for ,pollub'@'pluton.pol.lublin.pl' =
PASSWORD(,pluton%2022');
```

W powyższym przykładzie dodatkowo utworzono użytkownika o nazwie *piotrm*, który będzie mógł podłączyć się do bazy tylko z komputera lokalnego opisanego zarówno adresem IP, jak i nazwą domenową. Z kolei polecenie `SET PASSWORD` nie tylko przypisuje hasło do użytkownika łączącego się z danego hosta, lecz także szyfruje to hasło w bazie danych, co dodatkowo podwyższy poziom bezpieczeństwa tych danych.

Kolejnym krokiem w procedurze definiowania dostępu do bazy danych będzie przydzielenie użytkownikowi odpowiednich uprawnień do wykonania operacji. Przyznawanie praw dostępu do działań na zasobach bazodanowych może odbywać się na czterech poziomach:

- globalnym – uprawnienia dotyczą wszystkich baz danych, utworzonych na danym serwerze;
- bazy danych – uprawnienia dotyczą wszystkich tabel znajdujących się w danej bazie danych;
- tabeli – uprawnienia dotyczą pojedynczej tabeli znajdującej się we wskazanej bazie danych,
- kolumny – uprawnienia dotyczą możliwości wykonania operacji z użyciem wskazanej kolumny danej tabeli.

Ponieważ większość osób korzystających z aplikacji bazodanowych wykonuje polecenia CRUD, tj. odczytuje dane, wstawia nowe dane lub je modyfikuje w jednej bazie danych, należy przydzielić im wyłącznie takie prawa, jakie bezpośrednio wynikają z ich roli jako klienta aplikacji. Oznacza to, iż w przypadku osób dokonujących jedynie filtrowania danych należy przyznać im wyłącznie uprawnienie do selekcji wierszy. Podobnie osoby wykonujące działania pobierania danych i ich modyfikacji danych powinny mieć tylko prawo do operacji `SELECT` i `UPDATE`. Dodatkowo, można zawęzić możliwość wykonania tych operacji tylko do wskazanych tabel tej samej bazy danych.

Przyjmując założenie, że użytkownik *pollub*, łącząc się z bazą danych z podanych uprzednio hostów, powinien mieć możliwość wykonania operacji selekcji danych, ich modyfikacji czy wstawiania do dowolnej tabeli w bazie danych *password_wallet*, należy wykonać polecenia przedstawione na listingu 4.7.

Listing 4.7. Nadawanie uprawnień do wykonania operacji CRUD we wszystkich tabelach bazy danych password_wallet

```
grant select, insert, update on password_wallet.* to 'pol-
lub'@'212.182.18.39' ;
grant select, insert, update on password_wallet.* to 'pol-
lub'@'212.182.16.104' ;
grant select, insert, update on password_wallet.* to 'pol-
lub'@'pluton.pol.lublin.pl' ;
```

Jeśli jednak uprawnienia użytkownika *pollub* mają dotyczyć tylko wybranej tabeli, np. *passwords*, wówczas konieczne będzie wykonanie poleceń umieszczonych w listingu 4.8.

Listing 4.8. Nadawanie uprawnień do wykonania operacji CRUD w tabeli users bazy danych password_wallet

```
grant select, insert, update on password_wallet.users to
'pollub'@'212.182.18.39' ;
grant select, insert, update on password_wallet.users to
'pollub'@'212.182.16.104' ;
grant select, insert, update on password_wallet.users to
'pollub'@'pluton.pol.lublin.pl' ;
```

W najbardziej skrajnym przypadku można ograniczyć prawa użytkownika *pollub* do wykonania wybranych operacji na pojedynczej kolumnie lub kolumnach tabeli. Przykład taki zobrazowano na listingu 4.9.

Listing 4.9. Nadawanie uprawnień do wykonania operacji SELECT i UPDATE na wybranych kolumnach tabeli users bazy danych password_wallet

```
grant select (name, surname, login), update (email) on
password_wallet.users to 'pollub'@'212.182.18.39' ;
grant select (name, surname, login), update (email) on
password_wallet.users to 'pollub'@'212.182.16.104', 'pol-
lub'@'pluton.pol.lublin.pl' ;
```

Na listingu 4.9 można zauważyć, iż możliwe jest użycie jednego polecenia `GRANT` do nadawania tych samych uprawnień jednemu lub wielu użytkownikom, co może stanowić pewnego rodzaju ułatwienie dla niektórych administratorów baz danych.

Zastosowana powyżej procedura nadawania użytkownikowi bazy danych uprawnień do wykonania operacji nie jest jednak efektywna z punktu widzenia zarządzania tymi uprawnieniami. Jeśli wyobrazimy sobie sytuację, w której określonej grupie użytkowników konieczne jest nadanie dodatkowego prawa, wówczas przedstawione dotychczas podejście wymagałoby wykonania polecenia `GRANT` dla każdego konta. Podobna sytuacja ma miejsce w przypadku wycofania uprawnienia grupie osób korzystających z zasobów bazodanowych. W takim przypadku pomocne staje się zdefiniowanie tzw. roli, czyli nazwanej grupy określonych uprawnień. Po utworzeniu roli należy przypisać do niej wymagane uprawnienia, a następnie przydzielić taką rolę określonemu użytkownikowi. Przykład tego rodzaju rozwiązania przedstawiono na listingu 4.10.

Listing 4.10. Tworzenie roli

```
create or replace role crud_passwall_users_role ;
grant select, insert, update on password_wallet.users to
crud_passwall_users_role ;
grant crud_passwall_users_role to 'pollub'@'212.182.18.39',
'pollub'@'212.182.16.104', 'pollub'@'pluton.pol.lublin.pl',
'piotrm'@localhost, 'piotrm'@'127.0.0.1' ;
```

Zaletą takiego podejścia jest wyeliminowanie konieczności przyznawania dodatkowych uprawnień bezpośrednio konkretnemu użytkownikowi, gdyż prawo takie przypisywane jest do roli, z której korzysta dana osoba. Jeśli w przyszłości zaistnieje konieczność zmiany uprawnień użytkownika, wówczas wystarczy jednym poleceniem przypisać lub usunąć takie uprawnienie z danej roli, którą przydzielono użytkownikowi, zamiast wielokrotnie powtarzać operację zmiany uprawnień dla każdej z osób.

Modyfikacja uprawnień dla użytkowników zalogowanych do bazy danych nie następuje automatycznie po wykonaniu polecenia `GRANT`. Konieczne jest w takiej sytuacji wykonanie polecenia:

```
FLUSH PRIVILEGES;
```

Rezultatem jego wykonania będzie ponowne odczytanie zawartości systemowej tabeli global_grants w celu aktualizacji praw użytkownika. Jeśli polecenie to nie zostanie wykonane, wówczas zmiana uprawnień przypisanych do konta nastąpi dopiero przy ponownym zalogowaniu się do bazy danych.

Zarządzanie uprawnieniami i rolami w bazie danych to nie tylko ich przypisywanie odpowiednio do roli czy konta, lecz także ich usuwanie. Do tego celu stosuje się polecenie `REVOKE`. Na listingu 4.11 zaprezentowano kod, który spowoduje wycofanie uprawnienia INSERT z roli crud_passwall_users_role, odebranie tej roli użytkownikowi piotrm logującemu się do bazy danych z komputera lokalnego oraz odebranie uprawnienia UPDATE do tabeli users użytkownikowi pollub, logującemu się do bazy password_wallet z hosta pluton.pol.lublin.pl.

Listing 4.11. Odebranie uprawnienia

```
revoke insert on password_wallet.users from
crud_passwall_users_role;
revoke crud_passwall_users_role from 'piotrm'@localhost,
'piotrm'@'127.0.0.1';
revoke update on password_wallet.users from 'pollub'@'plu-
ton.pol.lublin.pl';
```

W przeciwieństwie do polecenia `GRANT`, rezultat wykonania polecenia `REVOKE` jest natychmiastowy, co oznacza, że użytkownik traci możliwość wykonania operacji, której dotyczyło dane uprawnienie bądź rola.

Odbierając użytkownikowi rolę należy pamiętać, że polecenie to dotyczy wyłącznie roli jako całości, a nie pojedynczych praw wchodzących w jej skład. Oznacza to, że użytkownik po odebraniu mu danej roli, może nadal wykonywać określone działanie, gdyż posiada prawo do niego przyznane mu bezpośrednio lub w ramach innej posiadanej roli.

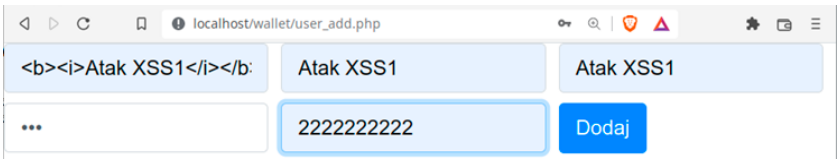
Podsumowując rozważania na temat zarządzania uprawnieniami użytkowników bazy danych należy podkreślić, iż nie jest zalecane przyznawanie uprawnień danemu użytkownikowi do wykonywania operacji we wszystkich bazach danych, znajdujących się na danym serwerze. Wysokie zagrożenie stanowi również sytuacja, w której użytkownik posiada pełne uprawnienia do wykonania operacji CRUD na wszystkich tabelach tej samej bazy danych. Dlatego też administrator bazy danych powinien z rozwagą przyznawać uprawnienia i okresowo monitorować operacje wykonywane w bazach danych na poszczególnych tabelach.

4.4. Ataki XSS

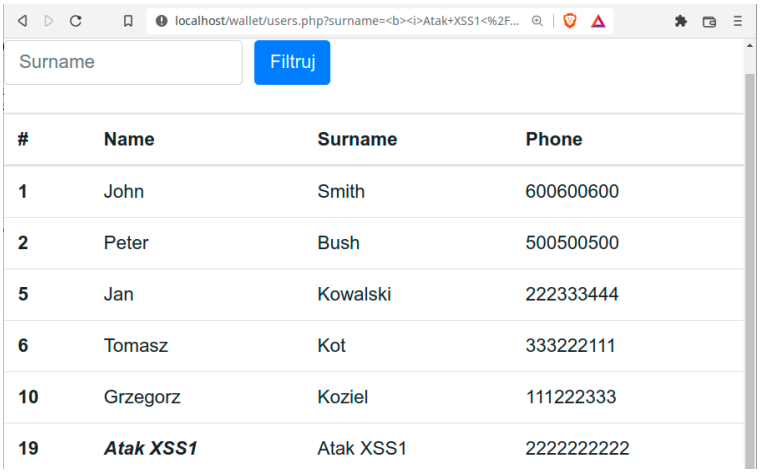
Ataki XSS (ang. Cross Site Scripting) polegają na umieszczeniu na stronie dodatkowego kodu. Ataki te określane są również jako wstrzyknięcie kodu. W przypadku ataków XSS wstrzykiwany jest kod HTML lub JavaScript. Rozróżniamy dwa typy ataków XSS:

- persistent XSS (stored XSS) – złośliwy kod jest na stałe umieszczany na stronie, zwykle poprzez wprowadzenie spreparowanych danych do bazy danych.
- reflected XSS (non-persistent XSS) – atak polegający na przekazaniu wybranemu użytkownikowi spreparowanych danych zawierających złośliwy kod. Rezultat ataku uzyskiwany jest natychmiast w odpowiedzi na żądanie użytkownika. Dane z żądania wpływają na wykonanie kodu strony wynikowej. Atak realizowany jest często poprzez dostarczenie użytkownikowi spreparowanego adresu URL [21].

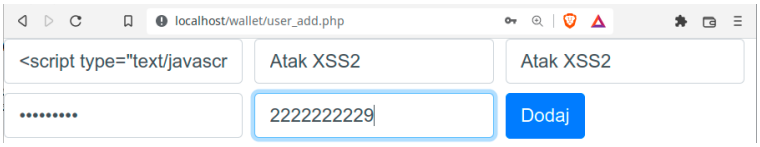
Przykładem prostego ataku XSS może być wstrzyknięcie kodu HTML, pokazanego na rysunku 4.5, powodującego zmianę wyglądu dołączonego tekstu, tak jak to zaprezentowano na rysunku 4.6. Taki atak narusza spójność wyglądu strony. O wiele groźniejsze są ataki wstrzyknięcia kodu JavaScript. Wstrzyknięty przez atakującego kod może się wykonywać na komputerach użytkowników aplikacji. Przykład takiego ataku został zaprezentowany na rysunku 4.7. Zamiast imienia dodawanego użytkownika podano fragment kodu JavaScript wyświetlającego okno dialogowe prezentujące zawartość ciasteczka witryny. Wstrzyknięty kod ma postać: `<script type="text/JavaScript"> alert(document.cookie);</script>`. Efekt ataku został przedstawiony na rysunku 4.8. Zanim zostanie wyświetlona witryna wyświetlane jest okno dialogowe zawierające identyfikator sesji. Obrazuje to powagę prezentowanego ataku. Zamiast wyświetlać identyfikator sesji atakujący może go przesłać do siebie i dokonać kradzieży sesji (ang. session hijack). Możemy również rozważyć nieco inną odmianę tego samego ataku. Wyobraźmy sobie, że atakujący umieścił instrukcję wyświetlania okna dialogowego w pętli nieskończonej. Uniemożliwi to wyświetlenie zawartości strony a na dodatek obciąży komputer klienta, negatywnie wpływając na jego pracę. Możliwe jest więc dokonanie w ten sposób ataku na dostępność aplikacji internetowej. Możliwe jest również umieszczanie dodatkowych elementów na atakowanej stronie. Na rysunku 4.9 pokazano efekt zamieszczenia elementu *div* zamiast wyświetlanego tekstu. Taki element może zawierać dowolną zawartość, również pobraną z innej strony WWW.



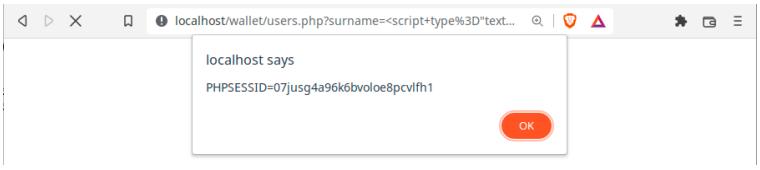
Rys. 4.5. Atak XSS przy użyciu ciągu „Atak XSS1”



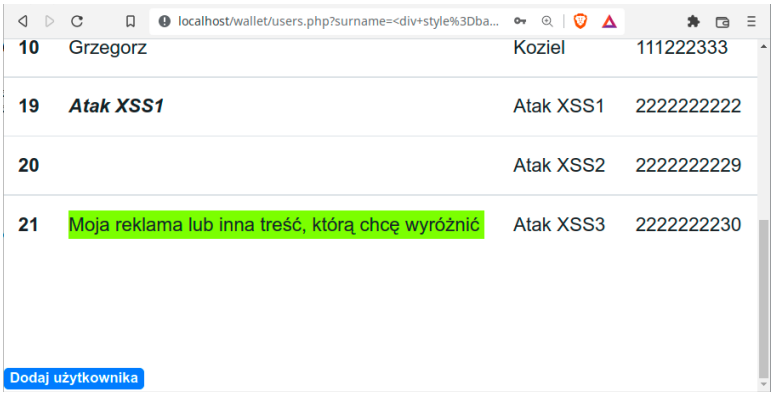
Rys. 4.6. Rezultat ataku XSS z rysunku 4.5



Rys. 4.7. Atak XSS przy użyciu ciągu „<script type=”text/JavaScript”> alert(document. cookie); </script>”



Rys. 4.8. Rezultat ataku XSS z rysunku 4.7



Rys. 4.9. Rezultat ataku XSS wykonanego poprzez użycie ciągu „<div style=background-color:GreenYellow;” Moja reklama lub inna treść, którą chcę wyróżnić ”

Zaprezentowane przykłady pokazują, że podatność na ataki XSS stanowi poważne zagrożenie aplikacji internetowej i może mieć poważne konsekwencje. Od zmiany zawartości strony przez wyciek danych użytkowników i umożliwienie włamania na ich konta po całkowitą utratę kontroli nad aplikacją i uniemożliwienie dostępu do niej. Dlatego też istotne jest skuteczne zabezpieczenie przed tego typu atakami.

Aby zabezpieczyć aplikację przed atakami XSS należy wszystkie dane wbudowywane w treść strony traktować jako niezaufane, niezależnie od ich pochodzenia. Mamy tu na myśli zarówno dane pobrane od użytkownika, jak i pochodzące z bazy danych, plików konfiguracyjnych, źródeł zewnętrznych czy innych zasobów. Dane powinniśmy filtrować pod kątem wystąpienia w nich sekwencji niebezpiecznych z punktu widzenia ataków XSS, usuwać z nich znaczniki oraz eskejpować znaki specjalne, czyli oznaczać je tak, by były traktowane jako zwykłe znaki tekstu.

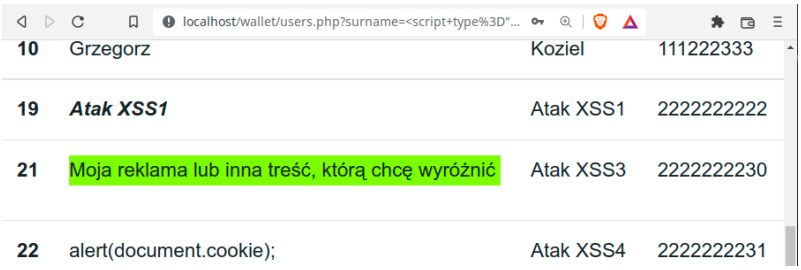
Usuwanie znaczników z kodu źródłowego może zostać wykonane za pomocą funkcji `strip_tags()`, do której jako parametr podajemy łańcuch znaków, z którego chcemy usunąć znaczniki. Zastosowanie funkcji `strip_tags()` omówione zostanie na przykładzie atakowanego formularza dodawania użytkowników. Dane z tego formularza przetwarzane są w funkcji `user_add()`. Dodamy więc na początku kodu tej funkcji wywołania instrukcji `strip_tags()` usuwające znaczniki z tekstu wprowadzonego przez użytkownika, tak jak to zaprezentowano na listingu 4.12.

Listing 4.12. Metoda user_add()

```
 public function user_add($surname, $name, $phone, $login,
  $password)
 {
  $surname=strip_tags($surname);
  $name=strip_tags($name);
  $phone=strip_tags($phone);
  $login=strip_tags($login);

  $password = "'" . $this->crypto->calculate_bcrypt(
   $password) . "'";
  $sql = „SELECT id from users where login=?";
…
//tu dalsza część kodu funkcji
```

Działanie funkcji `strip_tags()` przetestowane zostało poprzez ponowne przeprowadzenie ataku zaprezentowanego na rysunku 4.7. Do formularza dodawania użytkownika wprowadzono dane zawierające kod JavaScript. Użytkownik został poprawnie dodany, jednak wszystkie wprowadzone znaczniki zostały

usunięte. Lista użytkowników zawierająca dane dodane w trakcie ataku została pokazana na rysunku 4.10. Kod JavaScript jest widoczny w kolumnie imienia, jednak ze względu na brak odpowiednich znaczników traktowany jest on jako zwykły tekst.

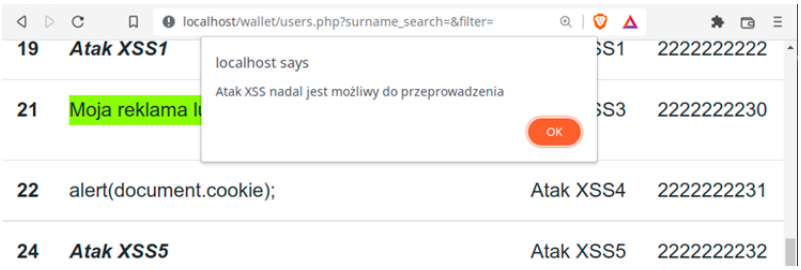


Rys. 4.10. Efekt ataku XSS po zastosowaniu funkcji strip_tags()

Funkcja `strip_tags()` nie sprawdzi się we wszystkich sytuacjach. Jeśli spróbujemy zamieścić na stronie tekst zawierający znaki „<" lub „>" – na przykład formułę matematyczną: „a<b/c" funkcja usunie wszystkie znaki począwszy od znaku „<", który zostanie uznany za początek znacznika.

Dodatkowym problemem jest fakt usuwania wszystkich znaczników z danych. Czasami wymagane jest zrobienie wyjątku i dozwolenie użycia wybranych znaczników. Taka możliwość została przewidziana przez twórców metody `strip_tags()`. Wymaga ona zastosowania białej listy znaczników. Przykładowo, aby zezwolić na użycie znaczników `<B>` oraz `<em>` funkcja `strip_tags()` powinna mieć postać: `strip_tags($napis, „<strong><em>");`.

Taka konstrukcja rzeczywiście zadziała. Niestety – nie w każdym przypadku prawidłowo. Możliwe jest przecież dodanie obsługi zdarzeń (czyli kodu JavaScript) do znaczników. Generuje to kolejny problem podatności na ataki XSS. Aby zobrazować problem, dodamy białe listy znaczników do funkcji `strip_tags()` używanej w naszej aplikacji. Zezwolimy na użycie znaczników `<em>` oraz `<b>` i zobaczymy co się stanie w przypadku wprowadzenia ciągu `<b><em onmouseover='alert(„Atak XSS nadal jest możliwy do przeprowadzenia")'>Atak XSS5</em></b>` zamiast imienia dodawanego użytkownika. Dane użytkownika zostaną dodane prawidłowo, wprowadzone imię będzie pogrubione, zgodnie z tym, na co zezwoliliśmy. Jednak po najechaniu na nie kursorem myszy wykona się kod JavaScript dodany do zdarzenia *onmouseover* znacznika `<em>`, co zostało zaprezentowane na rysunku 4.11. Aby zastosowana funkcja działała poprawnie konieczne jest dodatkowe zabezpieczenie przed wykonaniem kodu JavaScript zagnieżdżonego wewnątrz znaczników.

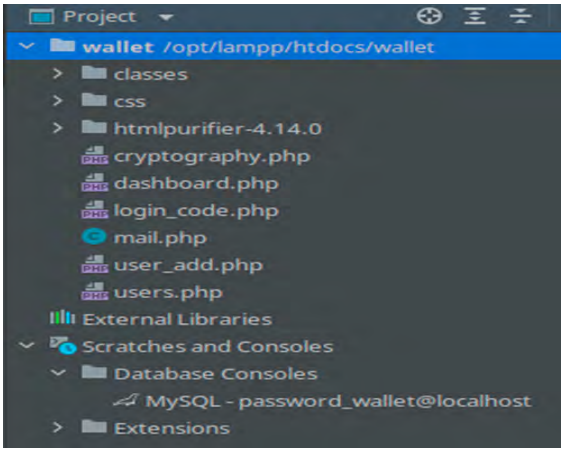


Rys. 4.11. Efekt ataku XSS po zastosowaniu białej listy znaczników

Rozwiązaniem wspomagającym jest eskejpowanie znaków specjalnych. Służy do tego funkcja `addslashes()`. Powoduje ona dodanie znaku ukośnika przed znakiem specjalnym znalezionym w przetwarzanym tekście. Odbiera mu to znaczenie specjalne. Funkcja ta pomocna jest podczas wprowadzania do bazy łańcuchów zawierających znak apostrofu (np. O'Neill, O'Reilly), lecz nie może być traktowana jako zabezpieczenie przed atakami XSS.

Rozwiązaniem, które jest rekomendowane do ochrony przed atakami XSS jest użycie zewnętrznej biblioteki dostarczanej przez znaną i poważaną organizację. Przykładami takich bibliotek są: HTML Purifier, Google Caja oraz OWASP AntiSamy. Dokonując wyboru biblioteki, należy zwrócić uwagę na to, by jej producent dostarczał bieżące aktualizacje.

Wykorzystanie zewnętrznego rozwiązania zostanie zaprezentowane na przykładzie biblioteki HTML Purifier. Jest ona dostępna na stronie http://html-purifier.org/ skąd pobieramy archiwum zawierające najnowszą wersję biblioteki. a następnie rozpakowujemy go do katalogu projektu by uzyskać strukturę plików i katalogów pokazaną na rysunku 4.12.



4.12. Struktura projektu zawierająca bibliotekę HTML Purifier

Biblioteki HTML Purifier użyjemy bezpośrednio w klasie *DB_PDO*. Dołączymy ją więc do tej klasy za pomocą instrukcji `require_once` a następnie utworzymy konfigurację, która jest wymagana do uruchomienia. W przykładzie użyjemy domyślnej konfiguracji. Przekazana zostanie ona jako parametr do konstruktora klasy *HTMLPurifier*. W efekcie otrzymujemy narzędzie gotowe do filtrowania dowolnych ciągów znaków pod kątem zabezpieczenia przed atakami XSS. Zastępujemy nim używaną dotychczas funkcję `strip_tags()`, tak jak to zaprezentowano na listingu 4.13.

Listing 4.13. Zastosowanie biblioteki HTML Purifier

```
<?php
require_once „crypto.php”;
require_once ,./htmlpurifier-4.14.0/library/HTMLPurifier.
auto.php’;

class DB_PDO
{
 private $db;
 private $crypto;
 private $config;
 private $purifier;

 //konstruktor
 public function __construct()
 {
  //dane połączenia bazą
  $servername = „localhost”;
  $username = „root”;
  $password = „”;
  $dbname = „password_wallet”;
  $dsn = “mysql:host=” . $servername . “;dbname=”
   . $dbname;
  try {
   //wywołanie konstruktora, nawiązanie połączenia z bazą
   $this->db = new PDO($dsn, $username, $password);
  } catch (PDOException $e) {
   //obsługa błędów
   print „Błąd połączenia z bazą!: <br/>”;
   die();
  }
  $this->crypto = new Crypto();
  $this->config = HTMLPurifier_Config::createDefault();
  $this->purifier = new HTMLPurifier($this->config);
```

```
 }

 //dodanie nowego użytkownika
 public function user_add($surname, $name, $phone, $login,
  $password)
 {
  $surname=$this->purifier->purify($surname);
  $name=$this->purifier->purify($name);
  $phone=$this->purifier->purify($phone);
  $login=$this->purifier->purify($login);
  $password = "'" . $this->crypto->calculate_bcrypt(
   $password) . "'";
  $sql = „SELECT id from users where login=?";
  ...
  //dalsza część kodu
 }
}
```

Skuteczność działania biblioteki HTML Purifier możemy zweryfikować poprzez przeprowadzenie ataków XSS na aplikację i zaobserwowanie efektów jej działania. Spróbujmy więc powtórzyć atak prezentowany na rysunku 4.11. Ponownie wstrzykujemy znacznik zawierający kod JavaScript. Następnie weryfikujemy, jak zachowuje się strona wynikowa. Efekty wcześniej przeprowadzonych ataków nadal są aktywne, lecz w przypadku weryfikowanego ataku nie obserwujemy żadnych jego skutków. Zerknijmy więc na kod strony. Na listingu 4.14 pokazano fragment strony wynikowej zawierający dwa ostatnie wiersze wyświetlanej tabeli. Pierwszy z nich pokazuje zawartość wiersza po ataku XSS prowadzonym wówczas, gdy do filtrowania używana była funkcja `strip_tags()`. Drugi wiersz zawiera treść umieszczoną w aplikacji po przeprowadzeniu identycznego ataku na aplikację, w której dane filtrowane były przez bibliotekę HTML Purifier. Widzimy, że znaczniki HTML pozostały w treści, natomiast kod JavaScript został w całości usunięty. Zabezpieczenie przed atakiem XSS zadziałało prawidłowo.

Listing 4.14. Zastosowanie biblioteki HTML Purifier

```
<tr>
 <th scope="row">24</th>
 <td><b><em onmouseover='alert(„Atak XSS nadal jest
  możliwy do przeprowadzenia")'>Atak XSS5</em></b></td>
 <td>Atak XSS5</td>
 <td>2222222232</td>
</tr>
<tr>
```

```
 <th scope=”row”>25</th>
 <td><b><em>Atak XSS6</em></b></td>
 <td>Atak XSS6</td>
 <td>222222233</td>
</tr>
```

4.5. Zadanie do samodzielnego wykonania

Zmodyfikuj zestaw uprawnień użytkownika bazodanowego tak, by posiadał on uprawnienia do wykonywania operacji CRUD na bazie *password_wallet*. Zastosuj zabezpieczenia przed atakami XSS na pozostałych stronach aplikacji przedstawionej w przykładzie.

5. Rejestrowanie aktywności użytkowników

Zastosowanie zabezpieczeń nie jest wystarczające do zapewnienia bezpieczeństwa aplikacji. Niezbędne są również mechanizmy, które pozwolą weryfikować, czy bezpieczeństwo jest zachowane oraz czy nie występują podejrzane zachowania wewnątrz chronionego systemu. Ponadto, każdy administrator powinien być przygotowany na to, że w końcu nastąpi udany atak na jego system. W tym przypadku konieczne jest zapewnienie możliwości zidentyfikowania, co się stało, jaki rodzaj ataku został zastosowany, w jaki sposób atakującemu udało się dostać do systemu oraz jaki był zakres ataku. Identyfikacja ta jest niezbędna do określenia rozmiaru szkód i potencjalnych sposobów ich naprawienia, a także do zidentyfikowania słabych punktów systemu w celu ich naprawienia i uniemożliwienia wystąpienia takiego samego ataku w przyszłości.

5.1. Rejestrowanie aktywności użytkownika

Mechanizmem istotnym z punktu widzenia bezpieczeństwa jest logowanie aktywności użytkownika. Pozwala na zapisanie informacji o czynnościach wykonywanych przez użytkowników w systemie. W niniejszej książce potraktujemy temat bardzo pobieżnie i skupimy się wyłącznie na rejestrowaniu faktu logowania się użytkowników. Pozwoli nam to na zaprezentowanie sposobu działania mechanizmu, lecz z całą pewnością nie wyczerpuje tematu. W rzeczywistym przypadku należy zastanowić się, co chcemy osiągnąć poprzez rejestrowanie aktywności użytkowników i dopiero na tej podstawie dobrać listę rejestrowanych danych.

Naszym celem będzie zarejestrowanie prób logowania użytkowników oraz adresów IP, z których użytkownicy logują się. Pozwoli to na wykrywanie potencjalnie niebezpiecznych adresów oraz kont, które mogą być celem ataku. Zabezpieczenia przedstawione w rozdziale 3.3 mogą okazać się niewystarczające w przypadku wyrafinowanych ataków prowadzonych przez doświadczonych hakerów. Dlatego też utrwalimy historię logowań użytkowników.

Informacje o logowaniach użytkowników umieścimy w tabeli *logins* naszej bazy danych. Skrypt tworzący tę tabelę zamieszczono na listingu 5.1.

Listing 5.1. Skrypt tworzący tabelę logins

```
CREATE TABLE logins (
  id          int(11) NOT NULL AUTO_INCREMENT,
  time        timestamp NOT NULL comment 'czas_logowania',
  successfull tinyint(1) NOT NULL,
```

```
  user_id     int(11) DEFAULT 1000000 NOT NULL,
  id_addres    bigint(20),
  login       varchar(100),
  PRIMARY KEY (id)) ;
ALTER TABLE logins modify column user_id int(11) DEFAULT
1000000 NOT NULL;
ALTER TABLE logins ADD CONSTRAINT FKlogins832101 FOREIGN
KEY (id_adres) REFERENCES ip_addesses (id) ON UPDATE Set
null ON DELETE Set null;
ALTER TABLE logins ADD CONSTRAINT FKlogins887312 FOREIGN
KEY (user_id) REFERENCES users (id) ON UPDATE Cascade;
```

Funkcjonalność rejestrowania prób logowania użytkownika wydzielimy do oddzielnej funkcji. Nazwiemy ją `user_login_register()`. Funkcja ta jako parametry będzie przyjmowała:

- *$correct_login* – przyjmujący wartość logiczną określającą, czy login podany przez użytkownika jest prawidłowy, czyli czy został odnaleziony w bazie danych,
- *$user* – przechowujący identyfikator logującego się użytkownika; w przypadku, gdy podany login nie został odnaleziony w bazie danych, zmienna ta będzie zawierała podany login,
- *$ip* – adres IP, z którego dokonano próby logowania,
- *$successfull* – będący zmienną logiczną określającą, czy próba logowania zakończyła się powodzeniem.

Funkcja `user_login_register()` będzie wprowadzała do tabeli logins informacje o próbach logowania użytkowników. Uwzględnimy w niej również przypadek, w którym użytkownik podaje niepoprawny login. Wówczas nie będzie możliwe wskazanie identyfikatora użytkownika, co uniemożliwi zapisanie rekordu w tabeli. Dlatego też do tabeli użytkowników dodano rekord użytkownika anonimowego o identyfikatorze 1000000, do którego przypisywane są wszystkie próby logowania dokonane z użyciem niepoprawnego loginu. Ze względów bezpieczeństwa konto użytkownika anonimowego jest skonfigurowane jako trwale zablokowane i nieaktywne. Kod funkcji rejestrującej próby logowania zaprezentowano na listingu 5.2.

Listing 5.2. Funkcja user_login_register()

```
public function user_login_register($user, $ip, $success-
full, $correct_login)
 {
  $time = date('Y-m-d H:i:s', time());
  if ($successfull) {
   //logowanie pomyślne
```

```
  $successfull = 1;
 } else
  $successfull = 0;
 if ($correct_login) {
  //login użytkownika poprawny
  $id = $user;
  $sql = „Insert into logins
    (time,successfull,user_id,id_addres) values
    (:time,:successfull,:id,:ip)”;
  $stmt = $this->db->prepare($sql);
  $stmt->bindValue(‘:time’, $time, PDO::PARAM_STR);
  $stmt->bindValue(‘:successfull’, $successfull,
    PDO::PARAM_BOOL);
  $stmt->bindValue(‘:id’, $id, PDO::PARAM_INT);
  $stmt->bindValue(‘:ip’, $ip, PDO::PARAM_STR);
 } else {
  $id = 1000000;
  $sql = „Insert into logins
    (time,successfull,user_id,id_addres,login) values
    (:time,:successfull,:id,:ip,:login)”;
  $stmt = $this->db->prepare($sql);
  $stmt->bindValue(‘:time’, $time, PDO::PARAM_STR);
  $stmt->bindValue(‘:successfull’, $successfull,
    PDO::PARAM_BOOL);
  $stmt->bindValue(‘:id’, $id, PDO::PARAM_INT);
  $stmt->bindValue(‘:ip’, $ip, PDO::PARAM_STR);
  $stmt->bindValue(‘:login’, $user, PDO::PARAM_STR);
 }
 $ile = $stmt->execute();
}
```

Aby rejestrowanie prób logowania działało, należy dodać wywołania funkcji `user_login_register()` do kodu aplikacji. Wywołania te należy umieścić wewnątrz funkcji `user_login_2F_step1()` oraz `user_login_2F_step2()` w momencie zakończenia procesu logowania, niezależnie od tego czy logowanie kończy się sukcesem, czy nie. Wówczas w bazie danych zarejestrowane zostaną wszystkie próby logowania.

Przedstawiony proces rejestrowania demonstruje jedynie sposób wykonania tego typu funkcjonalności. W rzeczywistym systemie należałoby rejestrować dodatkowe dane. Przydatne mogą być informacje o przyczynie niepowodzenia logowania czy też adresie IP, z którego dokonano próby logowania. Zestaw utrwalanych danych należy dobrać stosownie od potrzeb.

Uwagi wymaga również miejsce utrwalenia danych. W przedstawionym przykładzie autorzy zapisali rejestr prób logowań do tej samej bazy danych, która jest używana do przechowywania danych aplikacji. Podejście to miało na celu zredukowanie stopnia skomplikowania kodu. W rzeczywistym rozwiązaniu logi aplikacji powinny zostać umieszczone w niezależnej bazie danych, która powinna działać na niezależnej maszynie. Mechanizm rejestrowania powinien posiadać jedynie prawo do zapisu w tejże bazie danych. Powodem takiego podejścia jest ryzyko wystąpienia ataku na aplikację. Powinniśmy założyć, że atakujący będzie w stanie uzyskać pełny dostęp do systemu oraz bazy danych. Wówczas będzie mógł usunąć rejestry świadczące o jego aktywności. Umieszczenie tychże rejestrów na niezależnej maszynie uniemożliwi atakującemu bezpośredni dostęp do nich. Ponadto zestaw uprawnień zredukowany do prawa *insert* nie pozwoli atakującemu na odczytanie zawartości logów i sprawdzenie, czy jego aktywność została zarejestrowana.

5.2. Rejestrowanie dostępu do funkcji

Informacją istotną z punktu widzenia bezpieczeństwa systemu jest weryfikowanie, kto uzyskiwał dostęp do poszczególnych funkcji systemu. Aby taką weryfikację umożliwić należy zapisać każdy fakt użycia funkcji przez użytkownika. Możemy tego dokonać wewnątrz każdej z funkcji, umieszczając w nich instrukcje zapisujące fakt ich wywołania. Również w tym przypadku dla uproszczenia posłużymy się dotychczasową bazą danych. Wszystkim, którzy realizują funkcjonalność logowania aktywności użytkowników w działających systemach rekomendujemy użycie niezależnej bazy danych, zgodnie z uwagami zawartymi w rozdziale 5.1.

Do przechowywania rejestrów użycia funkcji utworzymy tabelę *functions_run*, która będzie przechowywała dane o użytkowniku wywołującym funkcję, nazwie użytej funkcji oraz parametrach jej wywołania. Skrypt tworzący tabelę *functions_run* został pokazany na listingu 5.3.

Listing 5.3 Skrypt tworzący tabelę functions_run

```
CREATE TABLE functions_run (
  id            int(11) NOT NULL,
  time          timestamp NOT NULL,
  user_id       int(11) DEFAULT 1000000 NOT NULL,
  function_name varchar(255) NOT NULL,
  parameters    text,
  PRIMARY KEY (id));
```

Funkcjonalność rejestrowania użytych funkcji wydzielimy do funkcji `function_usage_register()`. Parametrami funkcji będą: nazwa wywoływanej

funkcji oraz parametry jej wywołania. Jako, że lista parametrów będzie miała różną liczbę elementów dla każdej z funkcji, parametry połączymy w jeden łańcuch rozdzielany znakiem „|". W ten sposób uzyskamy możliwość zarejestrowania dowolnej liczby parametrów wywołania funkcji. Kod funkcji `function_usage_register()` został zaprezentowany na listingu 5.4.

Listing 5.4 Funkcja function_usage_register()

```
public function function_usage_register($function,$parame-
ters)
 {
  echo „USER LOGIN REGISTER</BR>";
  echo "USER ID: ".$_SESSION['user_id']."</BR>";
  $time = date('Y-m-d H:i:s', time());
  $sql = „Insert into functions_run
   (time,user_id,function_name,parameters) values
   (:time,:id,:function_name,:parameters)";
  $stmt = $this->db->prepare($sql);
  $stmt->bindValue(':time', $time, PDO::PARAM_STR);
  $stmt->bindValue(':id', $_SESSION['user_id'],
   PDO::PARAM_INT);
  $stmt->bindValue(':function_name', $function,
   PDO::PARAM_STR);
  $stmt->bindValue(':parameters', $parameters,
   PDO::PARAM_STR);
  try{
   $ile = $stmt->execute();
  }catch(PDOException $ex){
   echo „Błąd rejestrowania użycia funkcji";
  }
 }
```

Przykładowe wywołanie funkcji `function_usage_register()` umieszczone wewnątrz funkcji `user_add()` zostało pokazane na listingu 5.5.

Listing 5.5 Rejestrowanie użycia funkcji user_add()

```
public function user_add($surname, $name, $phone, $login,
$password)
 {
  $params=$surname."|". $name."|". $phone."|". $login;
  $this→function_usage_register("user_add",$params);
  …
```

Przykładowa zawartość tabeli *functions_run* została zaprezentowana na rysunku 5.1. Widzimy na niej, że użytkownik o identyfikatorze 28 wyświetlał listę użytkowników, filtrował ją, dodał nowe konto użytkownika, a także próbował ataku wstrzyknięcia kodu SQL. Na podstawie tego typu rejestrów jesteśmy w stanie wykryć podejrzane zachowania użytkowników, a także często możemy wykryć próby ataków.

id	time	user_id	function_name	parameters
14	2022-05-12 12:14:13	28	get_users	Kot
15	2022-05-12 12:14:52	28	get_users	
16	2022-05-12 12:15:32	28	user_add	Mikołaj|Rej|123456789|mrej
17	2022-05-12 12:15:32	28	get_users	*NULL*
18	2022-05-12 12:21:31	28	get_users	Kot' union SELECT 1,1,1,1,1,1,1,1,1,1,1,1,1,1 #

Rys. 5.1. Tabela z rejestrem użycia funkcji

5.3. Zadanie do samodzielnego wykonania

Utwórz dodatkową bazę danych i użyj jej do zapisywania aktywności użytkowników. Oprócz rejestrowania logowań użytkowników dodaj rejestrowanie czasu wylogowania.

Dodaj do aplikacji funkcjonalność rejestrowania zmian w danych wprowadzanych przez użytkowników. W bazie przechowującej logi utwórz dodatkową tabelę i zapisuj w niej identyfikator użytkownika dokonującego zmian, nazwę tabeli bazodanowej, której zawartość uległa zmianie, dotychczasowy zestaw danych, nowe dane zapisywane w rekordzie oraz czas wprowadzenia zmian.

6. Autoryzacja użytkownika

Autoryzacja jest procesem określenia, czy dany podmiot ma uprawnienia do dostępu do wybranego zasobu. Przez zasoby rozumiemy tu zbiory danych oraz funkcje systemu. W niniejszym rozdziale zajmiemy się zagadnieniami określania tożsamości użytkownika, definiowania ról i uprawnień oraz autoryzacji użytkownika, czyli sprawdzenia, czy posiada on uprawnienia do wykonania poszczególnych funkcji.

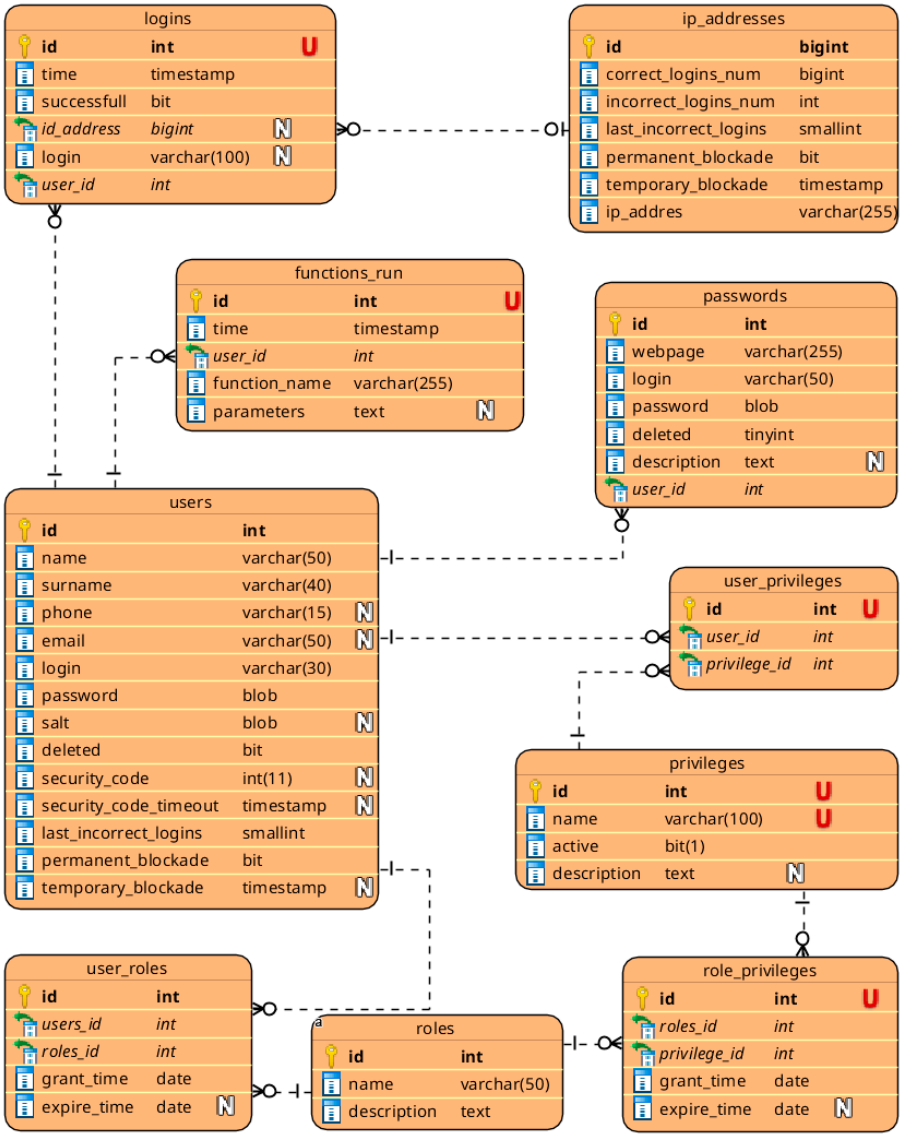
Jednym ze sposobów definiowania uprawnień jest utworzenie ich listy a następnie weryfikowanie, podczas wywoływania poszczególnych funkcjonalności, czy użytkownik ma nadane uprawnienie pozwalające na uzyskanie dostępu. Struktury danych pozwalające na realizację uprawnień przechowywane są zazwyczaj w bazie danych. Listę uprawnień przechowujemy wówczas w tabeli słownikowej. Informacja o nadanych uprawnieniach przechowywana jest w postaci zbioru danych zawierającego identyfikator użytkownika, któremu nadawane jest uprawnienie oraz identyfikatora przyznawanego uprawnienia. Posługując się językiem baz danych możemy powiedzieć, że jest to relacja wiele do wielu pomiędzy tabelą użytkowników i tabelą zawierającą listę uprawnień. Umieszczenie wpisu w tabeli realizującej tę relację jest jednoznaczne z nadaniem uprawnienia.

Definiowanie uprawnień dla każdego z użytkowników niezależnie daje duże możliwości konfiguracji zestawów uprawnień, jednak jest pracochłonne. Aby uprościć proces możemy zdefiniować role w systemie. Każdej roli nadamy określony zestaw uprawnień. Wszyscy użytkownicy posiadający daną rolę będą otrzymywali uprawnienia do niej przypisane. Takie podejście pozwala na szybkie przypisywanie zestawów uprawnień. Wpisuje się ono również w logikę biznesową aplikacji, bowiem często występuje w nich wielu użytkowników pełniących te same funkcje (mających tę samą rolę). Podejście to wymaga zdefiniowania listy ról. Do tego posłuży tabela słownikowa. Uprawnienia przypisane do poszczególnych ról zapisujemy tak samo, jak to miało miejsce w przypadku definiowania uprawnień użytkowników – przy użyciu tabeli realizującej relację wiele do wielu pomiędzy tabelą ról i tabelą uprawnień.

Pozostało jeszcze określenie sposobu przypisania ról użytkownikom. Tu również korzystamy ze znanego nam rozwiązania. Dodajemy tabelę realizującą relację wiele do wielu pomiędzy tabelami użytkowników i ról. W tabeli tej umieścimy identyfikatory użytkowników oraz przypisanych im ról. Taka organizacja struktury danych pozwoli na nadawanie użytkownikom wielu ról.

6.1. Struktury danych

W niniejszym podrozdziale przedstawiono przykładową strukturę bazy danych pozwalającą na przechowanie informacji o uprawnieniach użytkowników i ich rolach. Baza ta będzie używana do przechowywania danych o uprawnieniach na potrzeby kodu prezentowanego w dalszej części rozdziału. Schemat ERD projektowanej bazy zaprezentowany został na rysunku 6.1.



Rys. 6.1. Schemat ERD bazy danych

Do istniejącej dotychczas struktury dołączono tabele:

- *privileges* – tabela słownikowa uprawnień; w niej przechowywane są informacje o uprawnieniach używanych w aplikacji,
- *user_privileges* – tabela przechowująca informacje o uprawnieniach przypisanych poszczególnym użytkownikom,
- *roles* – tabela słownikowa przechowująca listę ról funkcjonujących w aplikacji,
- *role_privileges* – tabela zawierająca listę uprawnień przypisanych do poszczególnych ról,
- *user_roles* – tabela do zapamiętania listy ról przypisanych poszczególnym użytkownikom.

Brakujące struktury danych mogą zostać wygenerowane za pomocą skryptu przedstawionego na listingu 6.1.

Listing 6.1. Skrypt generujący struktury danych służące do przechowywania informacji o uprawnieniach użytkowników

```
CREATE TABLE privileges (
  id          int(11) NOT NULL,
  name        varchar(100) NOT NULL UNIQUE,
  active      tinyint(1) DEFAULT true NOT NULL,
  description text,
  PRIMARY KEY (id));
CREATE TABLE role_privileges (
  id           int(11) NOT NULL,
  roles_id     int(11) NOT NULL,
  privilege_id int(11) NOT NULL,
  grant_time   date NOT NULL,
  expire_time  date,
  PRIMARY KEY (id));
CREATE TABLE roles (
  id          int(11) NOT NULL,
  name        varchar(50) NOT NULL UNIQUE,
  description text,
  PRIMARY KEY (id));
CREATE TABLE user_privileges (
  id           int(11) NOT NULL,
  user_id      int(11) NOT NULL,
  privilege_id int(11) NOT NULL,
  PRIMARY KEY (id));
CREATE TABLE user_roles (
  id          int(11) NOT NULL,
  users_id    int(11) NOT NULL,
  roles_id    int(11) NOT NULL,
```

```
  grant_time  date NOT NULL,
  expire_time date,
  PRIMARY KEY (id));

ALTER TABLE user_privileges ADD CONSTRAINT FKuser_
privi756777 FOREIGN KEY (privilege_id) REFERENCES privi-
leges (id) ON UPDATE Cascade ON DELETE Restrict;
ALTER TABLE user_roles ADD CONSTRAINT FKuser_roles765422
FOREIGN KEY (roles_id) REFERENCES roles (id) ON UPDATE
Cascade ON DELETE Cascade;
ALTER TABLE user_roles ADD CONSTRAINT FKuser_roles761681
FOREIGN KEY (users_id) REFERENCES users (id) ON UPDATE
Cascade ON DELETE Cascade;
ALTER TABLE user_privileges ADD CONSTRAINT FKuser_
privi806618 FOREIGN KEY (user_id) REFERENCES users (id) ON
UPDATE Cascade ON DELETE Cascade;
ALTER TABLE role_privileges ADD CONSTRAINT FKrole_
privi595186 FOREIGN KEY (privilege_id) REFERENCES privi-
leges (id) ON UPDATE Cascade ON DELETE Cascade;
ALTER TABLE role_privileges ADD CONSTRAINT FKrole_
privi383001 FOREIGN KEY (roles_id) REFERENCES roles (id) ON
UPDATE Cascade ON DELETE Cascade;
```

6.2. Weryfikacja tożsamości na podstawie sesji

W rozdziale trzecim, przy omawianiu zagadnień dotyczących autentykacji użytkownika, użyty został mechanizm sesji do przechowania stanu aplikacji. Mechanizmu tego użyjemy również podczas autoryzacji użytkownika. Zgodnie z przyjętą w rozdziale trzecim logiką, sesja przechowuje dane użytkownika, łącznie z jego identyfikatorem. Identyfikatorem tym będziemy się posługiwać w celu jednoznacznego określenia tożsamości użytkownika. W kodzie każdej strony wchodzącej w skład naszej aplikacji umieścimy instrukcję `session_start()`. Spowoduje ona utworzenie sesji lub wznowienie istniejącej. Pozwoli to na odczytanie danych sesji, które są przechowywane po stronie serwera.

Aby zaprezentować działanie tego mechanizmu dodamy do strony users.php akapit, w którym wyświetlimy informację o loginie użytkownika lub informację o tym, że użytkownik nie jest zalogowany. Kod realizujący tę funkcjonalność został przedstawiony na listingu 6.2. Znajdziemy w nim instrukcję warunkową, która weryfikuje dwa warunki: sprawdza, czy istnieje tablica asocjacyjna *_SESSION* oraz czy kluczowi *logged_in* w tej tablicy przypisano wartość „logged_in". Tablica

_SESSION będzie istniała tylko wtedy, gdy sesja użytkownika będzie aktywna. Wartość w komórce o kluczu *logged_in* informuje nas o tym, czy użytkownik jest zalogowany czy nie – zgodnie z logiką przedstawioną w rozdziale trzecim. Jeśli użytkownik jest zalogowany, wówczas możemy skorzystać z danych zapisanych w sesji, w tym również z loginu użytkownika zapisanego w tablicy *_SESSION*. Należy jednak pamiętać, że dane te nie pojawiają się w sesji samoistnie. Za ich umieszczenie odpowiada programista. W kodzie z listingu 6.2 wykorzystujemy dane umieszczone w tablicy sesji przez funkcje prezentowane w rozdziale trzecim.

Listing 6.2. Kod wyświetlający login użytkownika

```
<?php
if (isset($_SESSION) && ($_SESSION[,logged_in'] == ,logged_
in')) {
 echo „<p>Jesteś zalogowany jako: <b>" . $_SESSION[,login']
. „</b></p></br>";
} else {
 echo „<p>Jesteś niezalogowany</p></br>";
}
?>
```

Zaprezentowany na listingu 6.2 przykład pokazuje, że mechanizm sesji może być użyty do zapamiętania danych i wykorzystywania ich na różnych stronach WWW wchodzących w skład tworzonej aplikacji. W tym rozdziale użyliśmy danych sesji do wyświetlenia loginu użytkownika. Jednak, z punktu widzenia identyfikacji zalogowanego użytkownika, bardziej użyteczne będzie wykorzystanie identyfikatora użytkownika, który również jest przechowywany w sesji w komórce o nazwie *user_id*.

6.3. Uprawnienia i role użytkowników

Strukturę danych przygotowaną w rozdziale 6.1 wykorzystamy do zdefiniowania ról i uprawnień użytkowników. Zacznijmy od zdefiniowania uprawnień. W naszym przykładzie wykorzystamy dwa uprawnienia: dodanie użytkownika oraz odczyt informacji o użytkownikach. Dodamy takie uprawnienia do tabeli *privileges* oraz nadamy uprawnienie odczytu informacji o użytkownikach jednemu z użytkowników naszej aplikacji. Polecenia SQL pozwalające na wykonanie tych czynności zostały przedstawione na listingu 6.3.

Listing 6.3. Zdefiniowanie uprawnień i przypisanie uprawnienia użytkownikowi

```
INSERT INTO `privileges`(`id`, `name`, `active`,
`description`)
VALUES
(1,'users read',1,'Users view'),
(2,'users create',1,'Users add');
INSERT INTO `user_privileges`(`id`, `user_id`, `privi-
lege_id`) VALUES (1,5,1);
```

Zwróćmy uwagę na pole *active* w tabeli *privileges*. Zostało ono użyte do oznaczenia, czy uprawnienie jest aktywne. Pozwala to na łatwe „dezaktywowanie" uprawnienia na poziomie całego systemu bez modyfikowania zestawów uprawnień przypisanych użytkownikom. Takie rozwiązanie jest przydatne w momencie, gdy czasowo chcemy wyłączyć z użycia jakąś funkcjonalność. Ustawienie uprawnienia jako nieaktywnego spowoduje, że żaden z użytkowników nie będzie mógł z niego korzystać, czyli moduły, do których wzmiankowane uprawnienie daje dostęp staną się niedostępne.

Oprócz uprawnień przypisywanych poszczególnym użytkownikom chcemy posłużyć się rolami. Zdefiniujmy więc dwie role o nazwach *admin* oraz *user*. Ponadto, określmy przysługujące im zestawy uprawnień. Niech użytkownikom z rolą *admin* przysługują oba zdefiniowane w aplikacji uprawnienia. Natomiast rola *user* będzie posiadała tylko uprawnienie do odczytu informacji o użytkownikach. Role zdefiniujemy dodając odpowiednie wpisy do tabeli *roles*. Natomiast przypisane im uprawnienia dodamy w tabeli *role_privileges*, tak jak to zaprezentowano na listingu 6.4. Zauważmy, że w tabeli *role_privileges* zostały zdefiniowane dwa pola przechowujące czas. Pozwalają one przechować informację o czasie nadania uprawnienia oraz czasie odebrania uprawnienia. Pola te mogą być użyteczne podczas analizy aktywności użytkowników. Możemy spotkać się z sytuacją, w której zapis aktywności użytkownika zawiera informację o wywołaniu funkcji, do której użytkownik nie ma uprawnień. Może to świadczyć o problemach z aplikacją lub być, po prostu, skutkiem odebrania uprawnień. W naszym przykładzie nie będziemy wykorzystywać tych pól.

Listing 6.4. Zdefiniowanie ról i przypisanie im uprawnień

```
INSERT INTO `roles`(`id`, `name`)
VALUES (1,'admin'),
(2,'user');
INSERT INTO `role_privileges`(`id`, `roles_id`, `privi-
lege_id`, `grant_time`)
VALUES (1,1,1,'2022_05_01'),
(2,2,1,'2022_05_01'),
(3,1,2,'2022_05_01');
```

Pozostało nam jeszcze przypisanie ról. Nadajmy dwóm wybranym użytkownikom role zapisane w bazie danych. Przykład nadania ról pokazano na listingu 6.5.

Listing 6.5. Przypisanie ról użytkownikom

```
INSERT INTO `user_roles`(`id`, `roles_id`, `users_id`,
`grant_time`)
VALUES (1,1,1,'2022-05-01'),
(2,2,2,'2022-05-01');
```

W tym momencie mamy już zdefiniowane i przypisane uprawnienia oraz role. Wykonaliśmy to za pomocą poleceń SQL. Pozwoliło to uniknąć budowania interfejsu graficznego do zarządzania rolami i uprawnieniami użytkowników. Możemy przystąpić do tworzenia modułu odczytującego uprawnienia użytkownika. W naszym przykładzie utworzymy funkcję, która będzie odczytywała listę uprawnień użytkownika i zapisywała ją w sesji. Takie podejście pozwoli nam uniknąć odpytywania bazy danych za każdym razem, gdy potrzebne będzie zweryfikowanie, jakie uprawnienia posiada użytkownik. Zmniejszymy więc obciążenie bazy danych i przyspieszymy działanie aplikacji. Rozwiązanie to ma jednak również wady. Zmiana uprawnień podczas trwania sesji nie spowoduje przeładowania zestawu uprawnień użytkownika. Konieczne będzie przelogowanie użytkownika lub zastosowanie innego mechanizmu ponownego odczytania uprawnień z bazy danych.

Funkcja odczytująca uprawnienia (nazwiemy ją `get_user_privileges()`) ma za zadanie odczytanie uprawnień przypisanych użytkownikowi. Następnie musi odczytać uprawnienia przypisane do wszystkich ról, jakie ten użytkownik posiada. Te dwa zestawy uprawnień muszą następnie zostać połączone w jeden a zduplikowane wpisy powinny zostać usunięte. Osiągniemy to konstruując odpowiednie zapytanie do bazy danych. Za pomocą instrukcji `union` połączymy dwa zapytania – jedno zwracające listę uprawnień przypisanych użytkownikowi a drugie zwracające listę uprawnień przypisanych rolom użytkownika. Pobraną listę zapiszemy w tablicy sesji, tworząc w niej tablicę o nazwie *privileges* zawierającą listę nazw uprawnień. Do tej tablicy będziemy się odwoływali za każdym razem, gdy będziemy weryfikowali listę uprawnień użytkownika. Kod funkcji `get_user_privileges()` został pokazany na listingu 6.6.

Listing 6.6. Kod funkcji get_user_privileges()

```
public function get_user_privileges($user_id)
{
 $this->function_usage_register("get_user_privileges",
  $user_id);
 try {
```

```
   //pobranie listy uprawnień użytkownika
   $sql = „select p.name from privileges p
    join role_privileges rp on p.id=rp.privilege_id
    join roles r on r.id=rp.roles_id
    join user_roles ur on r.id=ur.roles_id
    where ur.users_id=:uid and p.active=1
    union
    select p.name from privileges p
    join user_privileges up on p.id=up.privilege_id
    where up.user_id=:uid and p.active=1”;
   $stmt = $this->db->prepare($sql);
   //dołączenie identyfikatora użytkownika do zapytania
   $stmt->bindValue(":uid”, $user_id, PDO::PARAM_INT);
   $stmt->execute();
   //wyczyszczenie uprawnień zapisanych w sesji (o ile
     jakieś były)
   $_SESSION[,privileges’] = null;
   //zapisanie listy uprawnień w sesji
   while ($row = $stmt->fetch()) {
    $_SESSION[‘privileges’][] = $row[‘name’];
   }
  } catch (PDOException $e) {
   //obsługa błędów
   print „Błąd komunikacji z bazą!<br/>”;
   die();
  }
 }
```

Do prezentowanej na listingu 6.6 funkcji możemy dodać funkcjonalność weryfikowania, czy uprawnienia nie zostały odebrane. Weryfikacja ta odbywa się poprzez sprawdzenie czy w polu *expire_time* nie został ustawiony czas odebrania uprawnienia. Ci, którzy zechcą rozbudować implementację o taką funkcjonalność muszą jednak pamiętać, że przeładowanie uprawnień nastąpi dopiero po ponownym wywołaniu funkcji `get_user_privileges()`. Oznacza to, że odebranie uprawnień zalogowanemu użytkownikowi będzie miało skutek dopiero po ponownym wczytaniu uprawnień do sesji użytkownika.

Aby uprawnienia zostały wczytane do sesji należy wywołać funkcję `get_user_privileges()`. Powinna ona zostać wywołana po pomyślnym zakończeniu procesu logowania użytkownika. Dlatego też wywołanie w postaci `$this→get_user_privileges($_SESSION[,user_id’]);` dodamy w funkcji `user_login_2F_step2()` po pomyślnym zweryfikowaniu poprawności i czasu ważności kodu jednorazowego.

6.4. Kontrola dostępu do funkcji

Skuteczna kontrola dostępu do danych oraz funkcji wymaga zaimplementowania tejże kontroli w dwóch miejscach. Przede wszystkim kontrolujemy uprawnienia po stronie serwera. Każdorazowo, w momencie wywołania funkcji weryfikujemy, czy użytkownik, który ją wywołuje ma odpowiednie uprawnienia. Jeśli nie, przerywamy działanie funkcji. Umieszczenie weryfikacji uprawnień po stronie serwera zabezpiecza aplikację przed próbami manipulacji uprawnieniami wykonywanymi przez atakującego. Musimy pamiętać, że atakujący może spróbować wysłać spreparowane żądanie do serwera. Prostym przykładem preparowania żądań może być zapamiętanie treści żądania wysyłanego przez innego użytkownika, a następnie wysłanie jego zmodyfikowanej kopii do serwera. Jeśli kontrola uprawnień nie będzie zaimplementowana po stronie serwera, nastąpi wykonanie funkcji.

Drugim miejscem, w którym kontrola uprawnień powinna zostać zaimplementowana jest interfejs użytkownika. Dobrą praktyką jest utajnienie przed użytkownikami tych informacji o aplikacji, które nie są im niezbędne. Dlatego też elementy aplikacji, do których użytkownik nie powinien mieć dostępu nie powinny zostać zawarte w interfejsie klienta. Ukrywanie, czy też dezaktywowanie elementów interfejsu nie jest dobrą praktyką.

Kontrolę dostępu przedstawimy na przykładzie funkcjonalności wyświetlania listy użytkowników oraz dodawania nowego użytkownika. Po stronie serwera funkcjonalności te realizowane są przez funkcje `get_users()` oraz `user_add()`. Kod zawarty wewnątrz tych funkcji powinien się wykonać tylko wtedy, gdy użytkownik posiada odpowiednie uprawnienia. Dlatego też cały kod znajdujący się wewnątrz nich (oprócz logowania faktu wywołania funkcji) umieszczamy wewnątrz instrukcji warunkowej weryfikującej posiadanie uprawnień. Weryfikacja posiadania uprawnień polega na sprawdzeniu, czy istnieje tablica *privileges* w tablicy sesji oraz czy znajduje się w niej wpis o posiadaniu właściwego uprawnienia. Sposób weryfikacji uprawnień zaprezentowano na listingu 6.7 na przykładzie funkcji `user_add()`;

Listing 6.7. Weryfikacja uprawnień w funkcji user_add()

```
public function user_add($surname, $name, $phone, $login,
$password)
 {
  $params = $surname . „|” . $name . „|” . $phone . „|” .
   $login;
  $this->function_usage_register(“user_add”, $params);
  if (isset($_SESSION[‘privileges’]) && in_array(
   „users create”, $_SESSION[,privileges’])) {
   …
```

```
   //istniejący kod funkcji
  } else
   echo „Brak uprawnień”;
 }
```

6.5. Weryfikacja uprawnień w interfejsie użytkownika

W rozdziale 6.3 zaimplementowaliśmy kontrolę dostępu do funkcji. Uprawnienia użytkownika są już weryfikowane, lecz powinniśmy również wprowadzić odpowiednią kontrolę w interfejsie użytkownika. Naszym celem jest nieudostępnianie użytkownikowi funkcjonalności, do których nie powinien mieć dostępu. W tym celu wprowadzimy dodatkową weryfikację uprawnień w interfejsie graficznym. Wyświetlane będą tylko te elementy, do których użytkownik ma uprawnienia. Sposób realizacji zaprezentowany został na listingu 6.8. Przedstawia on plik users.php, do którego wprowadzono weryfikację czy użytkownik posiada uprawnienia do wyświetlenia listy użytkowników. Jeśli tak to lista jest wyświetlana. W przeciwnym wypadku wyświetlane jest stosowne powiadomienie. Analogicznie postąpiono w przypadku przycisku *dodaj użytkownika*. Jest on wyświetlany tylko wówczas, jeśli użytkownik posiada uprawnienie *user create*.

Listing 6.8. Wprowadzenie kontroli uprawnień na poziomie interfejsu użytkownika

```
<?php
session_start();
...
//istniejący kod
?>
<!DOCTYPE HTML>
<html lang=”pl”>
<head>
 ...  <!--istniejący kod-->
</head>
<body>
<header></header>
<?php
if (isset($_SESSION) && ($_SESSION[,logged_in’] == ,logged_
in’)) {
 echo „<p>Jesteś zalogowany jako: <b>” . $_SESSION[,login’]
. „</b></p></br>”;
} else {
 echo „<p>Jesteś niezalogowany</p></br>”;
}
```

```
//Weryfikacja czy użytkownik ma uprawnienia do wyświetlenia
listy kont
if (isset($_SESSION['privileges']) && in_array("users
read", $_SESSION['privileges'])) {
 ?>
    <section>
  ...
        <!-- kod wyświetlający listę użytkowników-->
  <?php
  //Weryfikacja czy użytkownik ma uprawnienia do tworzenia
   nowych kont
  if (isset($_SESSION['privileges']) && in_array(
   „users create", $_SESSION[,privileges'])) {
   echo ,<a href="user_add.php" class="badge
    badge-primary">Dodaj użytkownika</a>';
  }
  ?>
    </section>
 <?php
} else {
 ?>
        <p> Nie masz uprawnień do wyświetlania tej strony</
p>
 <?php
}
?>
<footer></footer>
</body>
</html>
```

6.6. Kontrola dostępu do danych

Kontrola dostępu do danych opiera się na kontroli dostępu do funkcji udostępniających dane oraz na weryfikacji praw własności do rekordu. Projektując strukturę danych umieszczamy identyfikator użytkownika, który jest właścicielem danych, wewnątrz rekordu. W momencie próby uzyskania dostępu do danych zwracamy użytkownikowi tylko te rekordy, które należą do niego. Zobrazujemy zagadnienie na przykładzie tabeli passwords. Nasza aplikacja ma przechowywać hasła wielu użytkowników, jednak niedopuszczalne jest udostępnienie haseł jednego użytkownika innemu. Dlatego też, realizując funkcjonalność wyświetlania haseł pobieramy tylko te hasła, które należą do aktualnie zalogowanego użytkownika.

Dodajmy do bazy danych dodatkowe uprawnienie, które będzie zezwalało na wyświetlenie własnych haseł i przypiszmy je do ról *user* oraz *admin*. Przykładowe instrukcje SQL zostały zawarte na listingu 6.9.

Listing 6.9. Dodanie uprawnienia wyświetlenia listy haseł

```
INSERT INTO `privileges`(`id`, `name`, `active`,
`description`)
VALUES
(3,'passwords read',1,'Passwords view');
INSERT INTO `role_privileges`(`id`, `roles_id`, `privi-
lege_id`, `grant_time`)
VALUES (4,1,3,'2022_05_01'),
(5,2,3,'2022_05_01');
```

Do pobierania haseł z bazy danych przeznaczymy funkcję `get_passwords()`, o postaci pokazanej na listingu 6.10. Funkcja ta będzie zwracała hasła użytkownika. Identyfikator użytkownika będzie pobierany przez funkcję wprost z tablicy sesji, aby zredukować ryzyko jego sfałszowania.

Listing 6.10. Pobranie haseł użytkownika z bazy danych

```
public function get_passwords()
 {
  $this->function_usage_register("get_passwords", "none");
  if (isset($_SESSION['privileges']) && in_array(
   „passwords read", $_SESSION[,privileges']])) {
   try {
    //skonstruowanie szablonu zapytania
    $sql = „SELECT * FROM passwords WHERE deleted=0
     and passwords.user_id LIKE ?";
    $stmt = $this->db->prepare($sql);
    //wiązanie danych z szablonem
    $stmt->bindValue(1, $_SESSION['user_id'],
     PDO::PARAM_INT);
    //wykonanie zapytania
    $stmt->execute();
    //zwrócenie wyników zapytania
    return $stmt;
   } catch (PDOException $e) {
    //obsługa błędów
    print „Błąd komunikacji z bazą!<br/>";
    die();
```

```
   }
  } else
   echo „Brak uprawnień”;
 }
```

Do wyświetlania haseł użytkownika stworzymy stronę passwords.php. Będzie ona pobierała hasła zwrócone przez funkcję `get_passwords()`, a następnie wyświetlała je. Kod strony passwords.php pokazano na listingu 6.11. Zwróćmy uwagę na to, że identyfikator użytkownika nie występuje w kodzie strony. Jest on zapisywany wewnątrz sesji użytkownika podczas uwierzytelniania. Z tego identyfikatora korzystamy, aby określić tożsamość użytkownika wyświetlającego swoje hasła.

Listing 6.11. Strona passwords.php

```
<?php
 session_start();
 include_once „classes/db_pdo.php”;
 $db = new DB_PDO();
?>
<!DOCTYPE HTML>
<html lang=”pl”>
<head>
 <meta http-equiv=”Content-Type” content=”text/html;
  charset=utf-8”/>
 <meta http-equiv=”Content-Language” content=”pl”/>
 <meta name=”description” content=””/>
 <meta name=”keywords” content=””/>
 <link rel=”stylesheet” href=”css/bootstrap.min.css”>
 <title>Portfel haseł</title>
</head>
<body>
<header></header>
<?php
 if (isset($_SESSION) && ($_SESSION[,logged_in’] ==
  ,logged_in’)) {
  echo „<p>Jesteś zalogowany jako: <b>” .
   $_SESSION[,login’] . „</b></p></br>”;
 } else {
  echo „<p>Jesteś niezalogowany</p></br>”;
 }
 if (isset($_SESSION[‘privileges’]) && in_array(“passwords
  read”, $_SESSION[,privileges’])) {
?>
```

```
  <section>
   <?php
   $stmt = $db->get_passwords();
   if ($row = $stmt->fetch()) {
   //zwrócono rekordy - wyświetlamy tabelę
   ?>
    <table class="table">
     <thead>
      <tr>
       <th scope="col">#</th>
       <th scope="col">Webpage</th>
       <th scope="col">Login</th>
       <th scope="col">Password</th>
      </tr>
     </thead>
     <tbody>
     <?php
     do {
     ?>
      <tr>
       <th scope="row"><?php echo $row["id"] ?></th>
       <td><?php echo $row["webpage"] ?></td>
       <td><?php echo $row["login"] ?></td>
       <td><?php echo $row["password"] ?></td>
      </tr>
    <?php
    } while ($row = $stmt->fetch()); //weryfikujemy czy
     jest kolejny rekord do pobrania
   }
   else
    echo „<p> Brak zapisanych haseł</p>"
   ?>
     </tbody>
    </table>
   </section>
<?php
} else {
?>
  <section>
   <p> Nie masz uprawnień do wyświetlania tej
    strony</p>
  </section>
 <?php
 }
 ?>
```

```
  <footer></footer>
 </body>
</html>
```

6.7. Zadanie do samodzielnego wykonania

Do istniejącego mechanizmu nadawania uprawnień dołącz zapamiętywanie czasów nadawania i odbierania uprawnień. Zaimplementuj funkcję, która na podstawie rejestru wywołania funkcji zweryfikuje czy wszyscy użytkownicy wywoływali tylko te funkcje, do których mają uprawnienia.

Bibliografia

1. Borso S., The Penetration Tester's Guide to Web Applications, 2019, Artech House
2. Das R., Johnson G., Testing and Securing Web Applications, 2020, Auerbach
3. Hoffmann A., Web Application Security, O'Reilly, 2020
4. Nabihah Ahmad, Lim Mei Wei, M. Hairol Jabbar, Advanced Encryption Standard with Galois Counter Mode, IOP Conf. Series: Journal of Physics: Conf. Series 1019 (2018) 012008 doi :10.1088/1742-6596/1019/1/012008, https://iopscience.iop.org/article/10.1088/1742-6596/1019/1/012008/pdf
5. Quinton É., Safety of Web Applications: Risks, Encryption and Handling Vulnerabilities with PHP, 2017, Elsevier
6. Shavers B., Bair J., Hiding Behind the Keyboard, Elsevier, 2016
7. Sroka K. (2016). Podpis elektroniczny a identyfikacja i uwierzytelnianie, Gorzów Wielkopolski, s. 109-110
8. Welling L., Thomson L., PHP i MySQL. Tworzenie stron WWW. Vademecum profesjonalisty. Wydanie V, Helion, 2017
9. https://hackernoon.com/cryptographic-hashing-c25da23609c3
10. https://project-rainbowcrack.com/table.htm
11. https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html
12. Cryptographic Storage Cheat Sheet, https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html
13. The Keyed-Hash Message Authentication Code (HMAC) , https://csrc.nist.gov/csrc/media/publications/fips/198/archive/2002-03-06/documents/fips-198a.pdf
14. session_start, https://www.php.net/manual/en/function.session-start.php
15. https://acrossecurity.com/papers/session_fixation.pdf
16. PHPMailer – A full-featured email creation and transfer class for PHP, https://github.com/PHPMailer/PHPMailer
17. https://www.w3schools.com/php/php_mysql_prepared_statements.asp
18. https://www.php.net/manual/en/book.pdo.php
19. https://www.w3schools.com/php/php_ref_filter.asp
20. https://medium.com/factory-mind/regex-tutorial-a-simple-cheatsheet-by-examples-649dc1c3f285
21. https://owasp.org/www-community/attacks/xss/
22. https://owasp.org/www-community/attacks/Password_Spraying_Attack

Indeks