Защита от XSS.

Санитизировал данные, поступавшие извне, функцией strip_tags().

Пример кода из index.php, санитизация кук:

```
$values = array();

$values['username'] = empty($_COOKIE['username_value']) ? '' :strip_tags($_COOKIE['username_value']);
```

….

При выгрузке данных пользователю из БД также санитизировал:

```
$row=$db->query("SELECT * FROM DBlab5 where login='".$_SESSION['login']."')->fetch();

$values['username'] =strip_tags($row['name']);
```

…..

Аналогично санитизировал данные при выводе их админу на admin.php.

Экранировал данные для админа функцией htmlspecialchars():

```
if (!empty($row)){

    echo "<p>Добро пожаловать: " . htmlspecialchars($_SERVER['PHP_AUTH_USER']) . "<br />"
```

….

Защита от SQL Injection.

Использовал подготовленные PDO запросы, например в index.php при загрузке данных нового пользователя в БД:

```
$db = new PDO('mysql:host=localhost;dbname=u20294', 'u20294', 5205554);

try {

    $stmt = $db->prepare("INSERT INTO DBlab5 (login, pass, name, mail, date, sex, limb, ability1, ability2, ability3, ability4, ability5, ability6, ability7, ability8, osebe) VALUES (:login, :pass, :name, :mail,  :date, :sex, :limb, :ability1, :ability2, :ability3, , :ability4, :ability5, :ability6, :ability7, :ability8, :osebe)");

    $stmt->bindParam(':login', $new_login);

    $stmt->bindParam(':pass', $new_pass);

    $stmt->bindParam(':name', $name);
```

```
    $stmt->bindParam(':mail', $mail);
    $stmt->bindParam(':date', $date);
    $stmt->bindParam(':sex', $sex);
    $stmt->bindParam(':limb', $limb);
    $stmt->bindParam(':ability1', $ability1);
    $stmt->bindParam(':ability2', $ability2);
    $stmt->bindParam(':ability3', $ability3);
    $stmt->bindParam(':ability4', $ability4);
    $stmt->bindParam(':ability5', $ability5);
    $stmt->bindParam(':ability6', $ability6);
    $stmt->bindParam(':ability7', $ability7);
    $stmt->bindParam(':ability8', $ability8);
    $stmt->bindParam(':osebe', $osebe);
    $stmt->execute();
}
catch(PDOException $e){}
$db = null;
```

Подготовленные запросы хороши тем, что не поддерживают мультизапросы, что не позволяет злоумышленнику выполнить собственный запрос, просто вставив его в середину изначального.

В версии нашего сервера такое ограничение распространяется и на функцию query(), однако на всякий случай используя ее я дополнительно экранировал данные от кавычек функцией quote(), пример из login.php:

```
$db = new PDO('mysql:host=localhost;dbname=u20294', 'u20294', 5205554);
$row=$db->query("SELECT login FROM DBlab5 where login=".$db->quote((string)$_POST['login'])." AND pass=".$db->quote((string)md5($_POST['pass'])))->fetch();
$db = null;
```

Защита от CSRF.

Такая уязвимость присутствовала на форме удаления пользователя на странице admin.php. В случае ее подделки с любого сайта на проект можно было бы отправить команду удаление юзера в обход страницы админа.

Для прикрытия уязвимости использовал систему токенов.

На admin.php добавил в уязвимую форму поле token для токена:

```
<form method='POST' action='delete.php'>
    <input type='hidden' name='token' value='cant_be_hacked' />
    <input type='submit' name='sendform' value='%s' />
</form>
```

Далее на странице delete.php добавил верификацию токена:

```
$token = $_POST['token'];
  if ($token=="cant_be_hacked"){
    $sth = $db->prepare("DELETE FROM DB7 WHERE login=:login");
    $sth->bindParam(':login', $login);
    $sth->execute();
  }
```

Таким образом формы со сторонних сайтов не будут удалять юзеров. Разумеется, в идеале токены должны генерироваться отдельно каждому юзеру для конкретной формы и сохранятся в БД на определенное время, однако такая реализация тянет на отдельное задание.

Защита от Include Upload.