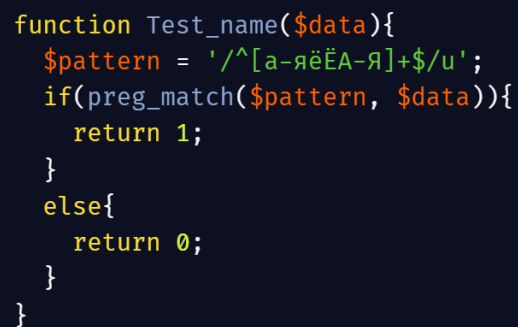


XSS

Для защиты формы заполнения данных в PHP от угрозы XSS, были использованы следующие меры безопасности:

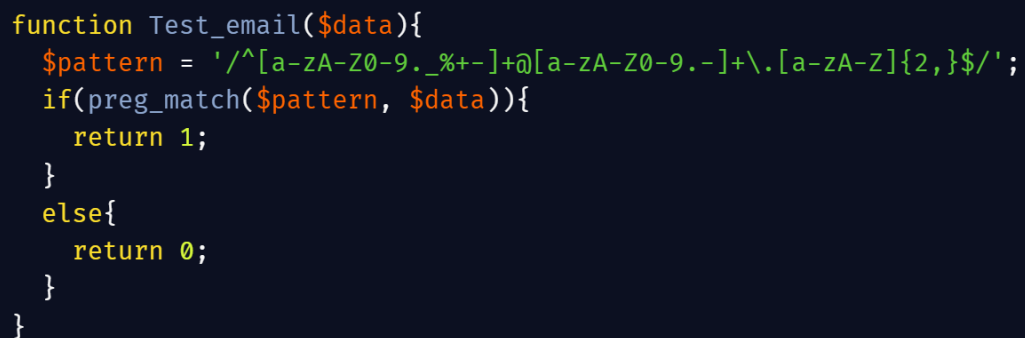
1. Валидация входных данных: Проверка входных данных на соответствие заданным правилам.

Некоторые примеры валидации из моего кода:



```
function Test_name($data){  
    $pattern = '/^[a-яёЁА-Я]+$/u';  
    if(preg_match($pattern, $data)){  
        return 1;  
    }  
    else{  
        return 0;  
    }  
}
```

Рис. 1. Функция для валидации имени



```
function Test_email($data){  
    $pattern = '/^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/';  
    if(preg_match($pattern, $data)){  
        return 1;  
    }  
    else{  
        return 0;  
    }  
}
```

Рис. 2. Функция для валидации email

```
function Test_biography($data){  
    $pattern = '/^[a-zA-Za-яA-Я\s.,!~-]{1,255}$/u';  
    if(preg_match($pattern, $data)){  
        return 1;  
    }  
    else{  
        return 0;  
    }  
}
```

Рис. 3. Функция для валидации поля "Биография"

SQL Injection

Для защиты формы заполнения данных в PHP от угрозы SQL Injection, были использованы следующие меры безопасности:

1. Использование подготовленных запросов: это позволяет отделить данные, введенные пользователем, от кода запроса.

Пример использования подготовленного запроса из моего кода:

```
$conn = new PDO('mysql:host=localhost;dbname=u52879', $user, $pass, [PDO::ATTR_PERSISTENT => true]);  
$stmt = $conn->prepare("INSERT INTO FORMS(name, email, year, gender, limbs, biography) VALUES (:name,  
:email, :year, :gender, :limbs, :biography)");  
$rez=$stmt->execute(['name'=>"$name", 'email'=>"$email", 'year'=>"$year", 'gender'=>"$gender",  
'limbs'=>"$kol", 'biography'=>"$biography"]);
```

Рис. 4. Пример использования подготовленного запроса

CSRF

Для защиты формы заполнения данных в PHP от угрозы CSRF, были использованы следующие меры безопасности:

1. Использование токена CSRF. Токен CSRF - это уникальная строка, которая генерируется на стороне сервера и передается вместе с формой на клиентскую сторону. Этот токен затем передается вместе с запросом на сервер для проверки.

Пример из моего кода:

```
$token = bin2hex(random_bytes(32));  
$token_time=time();  
$conn = new PDO('mysql:host=localhost;dbname=u52879', $user, $pass, [PDO::ATTR_PERSISTENT =>  
true]);  
$stmt = $conn->prepare("REPLACE INTO TOKENS(token, token_time) VALUES (:token, :token_time)");  
$rez_time=$stmt->execute(['token'=>"$token", 'token_time'=>"$token_time"]);
```

Рис. 5. Создание токена и запись его в таблицу на сервере

```
$token = $_POST['token'];  
$token_time = $_POST['token_time'];  
$conn = new PDO('mysql:host=localhost;dbname=u52879', $user, $pass, [PDO::ATTR_PERSISTENT =>  
true]);  
$stmt = $conn->prepare("SELECT * FROM TOKENS where token=:token");  
$stmt->execute(['token'=>"$token"]);  
$result_token = $stmt->fetchAll();  
if($result_token[0]['token']!= $token || $result_token[0]['token_time']-time()>60*60){  
    echo "Что-то пошло не так. Попробуйте ещё раз.";  
    exit();  
}
```

Рис. 6. Проверка токена полученного из формы

Include и Upload

Для защиты формы заполнения данных в PHP от Include и Upload уязвимости, были использованы следующие меры безопасности:

1. Используется полный путь к файлу вместо относительного пути при использовании функций include, чтобы исключить возможность подмены пути к файлу.

Пример из моего кода:

```
include(dirname(__FILE__) . '/form.php');
```

Рис. 7. Использование полного пути к файлу