

|  |  |
| --- | --- |
| МИНОБРНАУКИ РОССИИ | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | |
| Институт кибербезопасности и цифровых технологий | |
| Кафедра КБ-3 «Разработка программных решений и системное программирование»   |  | | --- | | **Отчёт по практике 4** | | **по дисциплине** «Разработка безопасного программного обеспечения» |     **Выполнила:**  Студентка 4 курса  Направление 02.03.02  Группа БФБО-01-20  Митрюхина Екатерина Андреевна  **Преподаватель**:  Поспелов П.М.  Москва, 2023 г. | |

1. Необходимо найти участок кода, содержащий инъекцию SQL кода в задании Blind Sql Injection на сайте dvwa.local с использованием статического анализатора кода (Можно использовать [официальный ресурс](https://github.com/digininja/DVWA) или виртуальную машину Web Security Dojo).

Воспользовалась статическим анализатором https://sonarsource.io/.

|  |
| --- |
| https://i.ibb.co/fr1nQnN/1.png |

Статический анализатор обнаружил 5 проблем.

1. $\_GET[‘id’]; - Использование GET запроса, а значит возможность атаки MITM(Man in the middle). Моя рекомендация: использовать POST запрос.
2. $id =
3. “SELECT first\_name, last\_name FROM users WHERE user\_id = ‘$id’;”; - Так как переменная ‘$id’ никак не обрабатывается и просто предается в SQL запрос, то здесь налицо возможность использования SQL инъекции. Моя рекомендация: обработать переменную ‘$id’.
4. $query =
5. mysqli\_query($GLOBALS[“\_\_mysqli\_ston”], $query);
6. Проанализировать код и сделать кодревью, указав слабые места

<?php

if( isset( $\_GET[ 'Submit' ] ) ) {

// Get input

$id = $\_GET[ 'id' ];

// Check database

$getid = "SELECT first\_name, last\_name FROM users WHERE user\_id = '$id';";

$result = mysqli\_query($GLOBALS["\_\_\_mysqli\_ston"], $getid ); // Removed 'or die' to suppress mysql errors

// Get results

$num = @mysqli\_num\_rows( $result ); // The '@' character suppresses errors

if( $num > 0 ) {

// Feedback for end user

$html .= '<pre>User ID exists in the database.</pre>';

}

else {

// User wasn't found, so the page wasn't!

header( $\_SERVER[ 'SERVER\_PROTOCOL' ] . ' 404 Not Found' );

// Feedback for end user

$html .= '<pre>User ID is MISSING from the database.</pre>';

}

((is\_null($\_\_\_mysqli\_res = mysqli\_close($GLOBALS["\_\_\_mysqli\_ston"]))) ? false : $\_\_\_mysqli\_res);

}

?>

**CWE-89:** Неправильная нейтрализация специальных элементов, используемых в команде SQL («SQL-инъекция»). *Описание уязвимости:* Продукт создает всю или часть команды SQL, используя внешние входные данные от вышестоящего компонента, но он не нейтрализует или неправильно нейтрализует специальные элементы, которые могут изменить предполагаемую команду SQL при ее отправке нижестоящему компоненту.

$getid = "SELECT first\_name, last\_name FROM users WHERE user\_id = '$id';";

**CWE-918:** Подделка запросов на стороне сервера (SSRF). Описание уязвимости: Веб-сервер получает URL-адрес или аналогичный запрос от вышестоящего компонента и извлекает содержимое этого URL-адреса, но недостаточно гарантирует, что запрос отправляется в ожидаемое место назначения.

Использование функций mysqli\_query и mysqli\_close с устаревшими методами работы с базой данных.

**CWE-703:** Неправильная проверка или обработка исключительных условий. Описание уязвимости: Продукт не предвидит и не справляется должным образом с исключительными условиями, которые редко возникают при нормальной работе продукта.

Использование or die и символа @.

**CWE-200:** Раскрытие конфиденциальной информации несанкционированному субъекту. Описание уязвимости: Продукт предоставляет конфиденциальную информацию субъекту, у которого нет явных полномочий на доступ к этой информации.

header( $\_SERVER[ 'SERVER\_PROTOCOL' ] . ' 404 Not Found' );

Исправление участка кода из dvwa.local (кодревью)

<?php

if( isset( $\_GET[ 'Submit' ]) ) {

$id = $\_GET[ 'id' ];

if (!preg\_match( '/[\s\'#]/', $input )) {

$query = $db->prepare( 'SELECT first\_name, last\_name FROM users WHERE user\_id = (:id) LIMIT 1;' );

$query->bindParam( ':id', mysqli\_real\_escape\_string($GLOBALS["\_\_\_mysqli\_ston"], $id ) );

$query->execute();

if( $num > 0 ) {

$html .= '<pre>User ID exists in the database.</pre>';

} else {

header( $\_SERVER[ 'SERVER\_PROTOCOL' ] . ' 404 Not Found' );

$html .= '<pre>User ID is MISSING from the database.</pre>';

}

}

}

?>

В данном случае я использовала подготавливаемые запросы, провела проектирование и отформатирование строк запроса, добавила проверку на ‘особые символы’.

1. Разработать свою систему вывода информации об объекте на любом языке, исключающий возможность инъекции SQL кода. Возможно исправление участка кода из dvwa.local  
   Требования к системе авторизации
   * Система вывода информации об объекте должна использовать запросы GET с параметрами, аналогичными из задания Blind SQL injection dvwa

Я разработала свою систему вывода информации об объекте (id) на языке Python. На данном веб-сайте есть форма для ввода id и если человек с данным id есть в базе данных, то выводится информация о нем (ФИО).

|  |
| --- |
|  |

Вот так выглядит выводимая информация о человеке, id которого есть в базе данных:

|  |
| --- |
|  |

Вот так выглядит страница с id, которого нет в базе данных:

|  |
| --- |
|  |

Для примера, если ввести SELECT \* FROM table WHERE id=1, то получим такой результат:

|  |
| --- |
|  |

Или введем 1 OR 1=1 --, ' OR 1=1--

|  |
| --- |
|  |

Как мы видим, «утечки» не происходит.

Вот код на python:

import sqlite3  
from flask import Flask, request, render\_template  
  
app = Flask(\_\_name\_\_)  
  
conn = sqlite3.connect('my\_data.db')  
cursor = conn.cursor()  
cursor.execute('''DROP TABLE IF EXISTS my\_data''')  
cursor.execute('''CREATE TABLE IF NOT EXISTS my\_data (id INTEGER PRIMARY KEY, first\_name TEXT, middle\_name TEXT, last\_name TEXT)''')  
cursor.execute('''INSERT INTO my\_data (first\_name, middle\_name, last\_name) VALUES   
 ('Иван', 'Иванович', 'Иванов'),  
 ('Федор', 'Федорович', 'Федоров')  
 ''')  
conn.commit()  
conn.close()  
  
@app.route('/')  
def index():  
 user\_info = None  
 message = None  
  
 user\_id = request.args.get('user\_id')  
  
 if user\_id:  
 user\_info = get\_user\_info(user\_id)  
  
 if user\_info:  
 message = None  
 else:  
 message = "Пользователь с данным id не найден"  
  
 return render\_template('site.html', user\_info=user\_info, message=message)  
  
def get\_user\_info(user\_id):  
 conn = sqlite3.connect('my\_data.db')  
 cursor = conn.cursor()  
 query = 'SELECT \* FROM my\_data WHERE id = ?'  
 cursor.execute(query, (user\_id,))  
 user\_info = cursor.fetchone()  
 conn.close()  
  
 return user\_info if user\_info else None  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(debug=True)

Я использую модули Flask и sqllite3. Защита от SQL-инъекций реализована в функции get\_user\_info таким образом, что вместо вставки значения `user\_id` напрямую в запрос, используется параметр `?`. Значение `user\_id` передается отдельно, что предотвращает возможность выполнения SQL-инъекций. Еще можно отметить то, что соединение с базой данных открывается в начале функции и закрывает соединение в конце.

Вот код на HTML:

<!DOCTYPE html>  
<html>  
<head>  
 <title>User information</title>  
</head>  
<body>  
 <h1>Информация о пользователе</h1>  
  
 <form method="get" action="/">  
 <label for="user\_id">Введите User ID:</label>  
 <input type="text" id="user\_id" name="user\_id" required>  
 <button type="submit">Search</button>  
 </form>  
 {% if message %}  
 <p>{{ message }}</p>  
 {% elif user\_info %}  
 <h2>Пользователь найден:</h2>  
 <p>ID: {{ user\_info[0] }}</p>  
 <p>Имя: {{ user\_info[1] }}</p>  
 <p>Отчество: {{ user\_info[2] }}</p>  
 <p>Фамилия: {{ user\_info[3] }}</p>  
 {% endif %}  
</body>  
</html>